BiDi Protocol

BiDi protocol allows users to create a bidirectional connection using WebSockets between the user-agent and the WebDriver. This is primarily to support two-way communication to allow user-agents to listen to the browser side events. This fits nicely with the event-driven nature of web browsers. BiDi Protocol enlists and explains the module, commands, events, and types required to perform a certain task using this protocol.

Currently, Selenium supports ChromeDevTools Protocol (CDP). CDP is browser version specific and supported only by Chromium-based browsers. Firefox provides partial support for CDP. For this reason, Selenium aims to provide support for a standardized set of APIs by supporting and keeping up with the BiDi Protocol. This aligns with the WebDriver support that Selenium currently provides.

Session creation based on capabilities

WebDriver uses sessions to carry on all interactions between the user-agent and the webdriver and in turn the web browser. Selenium implementation adapts to the same model and a Selenium webdriver instance lifecycle is tied to the session lifecycle. When a Selenium webdriver instance is created, a new WebDriver session is created. Following the same model, a BiDi session can be also attached to the same Selenium webdriver instance lifecycle.

From an end-user perspective, the Selenium BiDi API usage will be similar to the WebDriver APIs. This will ensure that the user is focused on what they want to achieve in terms of browser automation and not be bogged down by implementation details.

When a WebDriver session is created, it returns a set of capabilities. These returned capabilities will help identify if the browser supports BiDi or CDP. If the browser supports both, then creating a BiDi connection will be given priority.

CDP session creation

For a CDP connection the following returned capabilities contain the WebSocket address:

- "goog:chromeOptions" -> “debuggerAddress”
- "ms:edgeOptions" -> “debuggerAddress”
- "moz:debuggerAddress"

Steps to create a CDP socket connection:

- Get URI from the capabilities listed above based on the browser name.
- Send an HTTP request to “/json/version” to the same URI.
- The response will contain the “webSocketDebuggerUrl” field that has the URI to create a websocket connection.

BiDi session creation

To enable BiDi, when a WebDriver session is created, the request capabilities need to send “webSocketUrl” as true. This will be interpreted by the WebDriver based on the BiDi protocol connection process and it will return a websocket address in the returned capabilities.

For a BiDi connection, the following returned capabilities contain the WebSocket address:

- “webSocketUrl”

Steps to create a BiDi socket connection:

- “webSocketUrl” capability directly provided the URI to create the socket connection

Why is CDP session creation mentioned if the tech spec aims at BiDi APIs?

Selenium currently implements CDP in the background for a set of user-facing BiDi APIs. Since BiDi spec is in progress and most browser vendors are yet to implement it, CDP is used to provide similar functionality. Ideally, transitioning from CDP to BiDi should be as simple as removing all the CDP-related code and substituting it with the BiDi related. However, such a breaking change is not as simple.

This is also because the CDP APIs currently present in Selenium are not uniform. Most of the language bindings need the user to explicitly create a CDP session/connection and then leverage it. If given this situation, the BiDi implementation is to be added, then the user will need to explicitly create a BiDi connection (we already have this pattern in the Java binding). This adds cognitive overhead to the user. To rectify this and allow supporting CDP until BiDi is fully in place (there will also be a case where some browser versions support only CDP, some support only BiDi, and some support both) this spec aims to leverage the returned capabilities and provide the appropriate protocol session.

The following sections cover APIs that align with the BiDi spec since that is what will be supported in the long term. In most cases, the same APIs are applicable for CDP as well. Just the underlying implementation changes based on what type of session it is.

Internal implementation and structuring of how to provide support for CDP and BiDi (till BiDi is fully supported) is upto the design patterns and good practices in each language binding.

Note: This can be further discussed in the Selenium TLC meetings. Mainly, regarding how and if we want to support CDP in the future. It might be difficult to support both CDP and BiDi via same set of APIs for strictly typed languages because of difference in return type. In such cases preference will given to support BiDi APIs.

Subscribing to an event

BiDi allows subscribing to an event of interest. If the browsing context is not mentioned, then the event is subscribed globally i.e. for all browsing contexts. Browsing context id can be obtained either when a new browsing context is created using BiDi create command or when getWindowsHandles is called. Refer to https://w3c.github.io/webdriver/#contexts for more details.

Event

BiDi defines an event that contains the event name and the event params which defines the mapping of event data to a type (think of this like a Java class with a set of properties).

From a Selenium standpoint, an event will contain

1. The event name that maps to BiDi defined event name.
2. A mapper that maps the BiDi response from JSON to the expected type.

Note: The same set of APIs will be extended to work with Realms in the future
https://w3c.github.io/webdriver-bidi/#command-session-subscribe

Note: This ideally should be a private API that will not be available to the users (this will depend on the languages implementing as well). It is for the language bindings to be at par and understand requirements needed before module specific events and commands are implemented. Languages can choose to use the existing mechanisms available for CDP. Consider the APIs below as guidelines.
BiDi module specific event and command APIs are for end-users and that should be consistent among languages.

Java

X indicates the type that will be returned when an event is generated.

```
void addEventListener(Event<X> event, Consumer<X> handler)
void addEventListener(String browsingContextId, Event<X> event, Consumer<X> handler)
void addEventListener(List<String> browsingContextIds, Event<X> event,
Consumer<X> handler)
```

C#

X indicates the type that will be returned when an event is generated.
// Above statement is no longer valid. Removing it removes the comment. Just waiting to ensure that the changes are appropriate.

```
void AddEventListener(Event event, EventHandler<EventArgs> handler)
void AddEventListener(string browsingContextId, Event event, EventHandler<EventArgs> handler)
void AddEventListener(List<string> browsingContextIds, Event event, EventHandler<EventArgs> handler)
```

Javascript

```
eventOptions = {
       eventHandlerDict: {event: handler}
       browsingContextIds: []
}
function addEventListener(eventOptions)
```

Python

```
add_event_listener(browsingContextIdList = None, event, event_handler)
```

Ruby

```
add_event_listener(browsing_context_ids: nil, event_handler)
```

Unsubscribing to an event

BiDi allows unsubscribing to an event or set of events, either globally or for a set of browsing contexts.
Unsubscribing would be used internally when session is closed.

Note: This is a private API that will not be available to the users. It is for the language bindings to be at par and understand requirements needed before module specific events and commands are implemented. Languages can choose to use the existing mechanisms available for CDP. Consider the APIs below as guidelines.
BiDi module specific event and command APIs are for end-users and that should be consistent among languages.

Java

```
void removeEventListener(List<Event<X>> event)
```

C#

```
void RemoveEventListener(List<Event> event)
```

Javascript

```
eventOptions = {
       event
}

removeEventListener(eventOptions)
```

Python

```
remove_event_listener(browsingContextIdList = None, event_list)
```

Ruby

```
remove_event_listener(browsing_context_ids: nil, event_list)
```

Sending a Command

A command represents an action that can be performed on the browser that may return a response.

From Selenium standpoint, a command will contain:

1. The command name that maps to BiDi defined command name.
2. Command parameters as map, that need to sent along with the command name.
3. Expected return type.
4. A mapper that maps the BiDi response from JSON to the expected type.

Note: This is a private API that will not be available to the users. It is for the language bindings to be at par and understand requirements needed before module specific events and commands are implemented. Consider the APIs below as guidelines.
BiDi module specific event and command APIs are for end-users and that should be consistent among languages. All APIs can be sync or async depending on the good practice of the language implementing it.

Java

X indicates return type

```
X sendBiDiCommand(Command<X> command)
```

C#

X indicates return type

Note: Ensure the API async/await-able for C#

```
X SendBiDiCommand(Command<X> command)
```

Javascript

```
async send (params)
```

Python

```
send_BiDi_command(command, timeout = 0)
async send_BiDi_command(command, timeout = 0)
```

Ruby

```
send_cmd(method, **params)
```

Log Module APIs

The Log module provides APIs to allow listening to any type of log events.
Typically, the log event can be either of the type:

1. Console
   OR
2. Javascript
   OR
3. Generic (if not console or Javascript)

These APIs will be placed under the LogInspector class.

Selenium provides user-friendly APIs for each of the specific log types. Additionally, one API allows the users to listen to every log irrespective of the type.
Note: The same set of APIs will be extended to work with Realms in the future
https://w3c.github.io/webdriver-bidi/#command-session-subscribe

Initializing Log Module

When a module is initialised, it will check if a BiDi or CDP session exists for the same, else it will create one.

Java

```
LogInspector logInspector = new LogInspector(driver);
LogInspector logInspector = new LogInspector(browserContextId, driver);
LogInspector logInspector = new LogInspector(browserContextIds, driver);
```

C#

yet to be implemented

```
LogInspector logInspector = new LogInspector(driver);
LogInspector logInspector = new LogInspector(browserContextId, driver);
LogInspector logInspector = new LogInspector(browserContextIdSet, driver);
```

Javascript

```
logInspector = new LogInspector(driver)
```

Python

yet to be implemented

```
logInspector = LogInspector(driver);
```

Ruby

```
log_inspector = LogInspector.new(driver)
```

Event APIs

Java

```
void onConsoleEntry(Consumer<ConsoleLogEntry> consumer)
void onConsoleEntry(Consumer<ConsoleLogEntry> consumer, FilterBy filter)

void onJavaScriptLog(Consumer<JavascriptLogEntry> consumer)
void onJavaScriptLog(Consumer<JavascriptLogEntry> consumer, FilterBy filter)

void onJavaScriptException(Consumer<JavascriptLogEntry> consumer)

void onGenericLog(Consumer<GenericLogEntry> consumer)
void onGenericLog(Consumer<GenericLogEntry> consumer, FilterBy filter)

void onLog(Consumer<LogEntry> consumer)
void onLog(Consumer<LogEntry> consumer, FilterBy filter)
```

C#

yet to be implemented

```
void OnConsoleLog(EventHandler<ConsoleLogEntryArgs> handler)

void OnJavaScriptLog(EventHandler<JavascriptLogEntryArgs> handler)

void OnJavaScriptException(EventHandler<JavascriptLogEntryArgs> handler)

void OnGenericLog(EventHandler<GenericLogEntryArgs> consumer)

void OnLog(EventHandler<LogEntryArgs> handler)
```

Javascript

```
onConsoleEntry(callback, filterBy = undefined) // callback : it is the handler

onJavaScriptLog(callback, filterBy = undefined)

onJavaScriptException(callback)

onLog(callback, filterBy = undefined)
```

Python

yet to be implemented

```
on_console_log(browsingContextIdList = None, eventHandler)

on_javascript_log(browsingContextIdList = None, eventHandler)

on_javascript_exception(browsingContextIdList = None, eventHandler)
```

```
on_generic_log(browsingContextIdList = None, eventHandler)

on_log(browsingContextIdList = None, eventHandler)
```

Ruby

```
on_console_entry(filter_by = nil, &block)  // &block : it is the handler
on_javascript_log(filter_by = nil, &block)
on_javascript_exception(&block)
on_log(filter_by = nil, &block)
```

Browsing Context Module APIs

The browsing context commands defined in WebDriver BiDi seem to be aligned towards life cycle of a browsing context. Each command in the browsing context module requires passing in a browsingContext id. The idea here to is map this to a BrowsingContext class. Every time a new object is created, a new browsing context is created and internally that browsing context's id is used for all commands.

Initializing Browsing Context Module

Java

```
BrowsingContext context = new BrowsingContext(driver, browsingContextId); // in case user wants to use existing
browsing window/tab to run browsing context specific commands
BrowsingContext context = new BrowsingContext(driver, type, referenceBrowsingContext);
BrowsingContext context = new BrowsingContext(driver, type);
```

C#

yet to be implemented

```
BrowsingContext context = new BrowsingContext(driver, browsingContextId); // in case user wants to use existing
browsing window/tab to run browsing context specific commands
BrowsingContext context = new BrowsingContext(driver, type);
BrowsingContext context = new BrowsingContext(driver, type, referenceBrowsingContext);
```

Javascript

```
const BrowsingContext = require('selenium-webdriver/bidi/browsingContext');

browsingContext = await BrowsingContext(driver, {browsingContextId: id})   // in case user wants to use existing
browsing window/tab to run browsing context specific commands

browsingContext = await BrowsingContext(driver, {type: value})    // value = 'window' or 'tab'

browsingContext = await BrowsingContext(driver, {type: value1, referenceContext: value2})
```

Python

yet to be implemented

```
browsing_context = BrowsingContext(driver, browsingContextId); // in case user wants to use existing browsing
window/tab to run browsing context specific commands
browsing_context = BrowsingContext(driver, type);
browsing_context = BrowsingContext(driver, type, referenceBrowsingContext;
```

Ruby

```
browsing_context = BrowsingContext.new(driver: , browsing_context_id: ) // in case user wants to use existing
browsing window/tab to run browsing context specific commands

browsing_context = BrowsingContext.new(driver: , type: )
browsing_context = BrowsingContext.new(driver: , type: , reference_context: )
```

Command APIs

Java

```
String captureScreenshot() // enum defining output type might be needed in the future. Support additional types
when the protocol supports additional types
https://w3c.github.io/webdriver-bidi/#command-browsingContext-captureScreenshot

void close() // Note: closing behaviour is yet to be defined [Issue #w3c/webdriver-bidi#170]. Accordingly
Selenium might need to close the associated session if needed.

private String create(BrowsingContextType type) // enum defining "tab" and "window"
private String create(BrowsingContextType type, String referenceBrowsingContext) // enum defining "tab" and
"window"

BrowsingContextInfoList getTree()
BrowsingContextInfoList getTree(int maxDepth)

BrowsingContextInfoList getTopLevelContexts()

void handleUserPrompt()
void handleUserPrompt(boolean accept, String userText)
void handleUserPrompt(boolean accept)
void handleUserPrompt(String userText)

NavigationResult navigate(String url)
NavigationResult navigate(String url, ReadinessState readinessState)
// enum ReadinessState defining "none" / "interactive" / "complete"

void reload()
void reload(boolean ignoreCache, ReadinessState readinessState) // enum ReadinessState defining "none" /
"interactive" / "complete"
void reload(boolean ignoreCache)
void reload(ReadinessState wait)
```

C#

yet to be implemented

```
String CaptureScreenshot() // Support additional types when the protocol supports additional types
https://w3c.github.io/webdriver-bidi/#command-browsingContext-captureScreenshot

void Close()

string Create(BrowsingContextType type) // enum defining "tab" and "window"
string Create(BrowsingContextType type, string referenceBrowsingContext) // enum defining "tab" and "window"

BrowsingContextInfoList GetTree()
BrowsingContextInfoList GetTree(int maxDepth)

static BrowsingContextInfoList getTopLevelContexts()

void HandleUserPrompt()
void HandleUserPrompt(bool accept, string userText)
void HandleUserPrompt(bool accept)
void HandleUserPrompt(string userText)

NavigationResult Navigate(String url)
NavigationResult Navigate(String url, ReadinessState wait) // enum ReadinessState defining "none" / "interactive"
/ "complete"

void Reload()
void Reload(bool ignoreCache, ReadinessState wait) // enum ReadinessState defining "none" / "interactive" /
"complete"

void Reload(bool ignoreCache)
void Reload(ReadinessState wait)
```

Javascript

```
captureScreenshot() // Support additional types when the protocol supports additional types
https://w3c.github.io/webdriver-bidi/#command-browsingContext-captureScreenshot // yet to be implemented


async close()

async create(type, referenceContext) // type: defining "tab" and "window"

async getTree(maxDepth = undefined)

static getTopLevelContexts() // yet to be implemented

handleUserPrompt(accept = true, String userText = null) // yet to be implemented

async navigate(url, readinessState = undefined) // object readinessState defining "none" / "interactive" /
"complete"

async printPage(options = {}) // options map contains user defined options for print page

reload(ignoreCache = false, readinessState[wait] = null) // object readinessState defining "none" / "interactive"
/ "complete" // yet to be implemented
```

Python

yet to be implemented

```
capture_screenshot() # returns screenshot as string
Support additional types when the protocol supports additional types
https://w3c.github.io/webdriver-bidi/#command-browsingContext-captureScreenshot

close()

create(browsingContextType[type]) # returns browsing context as string, browsingContextType is enum defining
"tab" and "window"

getTree(maxDepth = None) # returns list of browsing context info as defined in
https://w3c.github.io/webdriver-bidi/#type-browsingContext-BrowsingContextInfo

@staticmethod
getTopLevelContexts()

handle_user_prompt(accept = None, userText = None)

navigate(URL url, wait = None) # wait is enum of ReadinessState defining "none" / "interactive" / "complete" ,
returns navigation result as defined in https://w3c.github.io/webdriver-bidi/#command-browsingContext-navigate

reload(ignoreCache = None, wait = None) # wait is enum of ReadinessState defining "none" / "interactive" /
"complete"
```

Ruby

```
Capture_screenshot    // Support additional types when the protocol supports additional types
https://w3c.github.io/webdriver-bidi/#command-browsingContext-captureScreenshot // yet to be implemented

close

create(type, reference_context) # returns browsing context as string, type = :tab or :window

get_tree(max_depth: nil)
self.get_top_level_contexts()  // yet to be implemented

handle_user_prompt(accept: true, user_text: nil)    // yet to be implemented
```

```
navigate(url: , readiness_state: nil) # readiness_state = "none" / "interactive" / "complete"

reload(ignore_cache: false, readiness_state[:wait]: nil) # object readiness_state defining "none" / "interactive"
/ "complete"    // yet to be implemented
```

Event APIs

yet to be implemented

Java

```
void onBrowsingContextCreated(Consumer<BrowsingContextInfo> consumer)

void onBrowsingContextDestroyed(Consumer<BrowsingContextInfo> consumer)

void onNavigationStarted(Consumer<NavigationInfo> consumer)

void onFragmentNavigated(Consumer<NavigationInfo> consumer)

void onDomContentLoaded(Consumer<NavigationInfo> consumer)

void onBrowsingContextLoaded(Consumer<NavigationInfo> consumer)

void onDownloadStarted(Consumer<NavigationInfo> consumer)

void onNavigationAborted(Consumer<NavigationInfo> consumer)

void onNavigationFailed(Consumer<NavigationInfo> consumer)

void onUserPromptClosed(Consumer<UserPromptInfo> consumer)

void onUserPromptOpened(Consumer<UserPromptInfo> consumer)
```

C#

```
void OnBrowsingContextCreated(EventHandler<BrowsingContextInfoArgs> handler)

void OnBrowsingContextDestroyed(EventHandler<BrowsingContextInfoArgs> handler)

void OnNavigationStarted(EventHandler<NavigationInfoArgs> handler)

void OnFragmentNavigated(EventHandler<NavigationInfoArgs> handler)

void OnDomContentLoaded(EventHandler<NavigationInfoArgs> handler)

void OnBrowsingContextLoaded(EventHandler<NavigationInfoArgs> handler)

void OnDownloadStarted(EventHandler<NavigationInfoArgs> handler)

void OnNavigationAborted(EventHandler<NavigationInfoArgs> handler)

void OnNavigationFailed(EventHandler<NavigationInfoArgs> handler)

void OnUserPromptClosed(EventHandler<UserPromptInfoArgs> handler)

void OnUserPromptOpened(EventHandler<UserPromptInfoArgs> handler)
```

Javascript

```
onBrowsingContextCreated(callback)

onBrowsingContextDestroyed(browsingContextIdList = null, eventHandler) // yet to be implemented
```

```
onNavigationStarted(browsingContextIdList = null, eventHandler) // yet to be implemented

onFragmentNavigated(browsingContextIdList = null, eventHandler) // yet to be implemented

onDomContentLoaded(callback)

onBrowsingContextLoaded(callback)

onDownloadStarted(browsingContextIdList = null, eventHandler) // yet to be implemented

onNavigationAborted(browsingContextIdList = null, eventHandler) // yet to be implemented

onNavigationFailed(browsingContextIdList = null, eventHandler) // yet to be implemented

onUserPromptClosed(browsingContextIdList = null, eventHandler) // yet to be implemented

onUserPromptOpened(browsingContextIdList = null, eventHandler) // yet to be implemented
```

Python

```
on_browsing_context_created(browsingContextIdList = None, eventHandler)

on_browsing_context_destroyed(browsingContextIdList = None, eventHandler)

on_navigation_started(browsingContextIdList = None, eventHandler)

on_fragment_navigated(browsingContextIdList = None, eventHandler)

on_dom_content_loaded(browsingContextIdList = None, eventHandler)

on_browsing_context_loaded(browsingContextIdList = None, eventHandler)

on_downloaded_started(browsingContextIdList = None, eventHandler)

on_navigation_aborted(browsingContextIdList = None, eventHandler)

on_navigation_failed(browsingContextIdList = None, eventHandler)

on_user_prompt_closed(browsingContextIdList = None, eventHandler)

on_user_prompt_opened(browsingContextIdList = None, eventHandler)
```

Ruby

*All the methods will have a ‘yield’ statement to execute the block (even_handler) with which they are called.

```
on_browsing_context_created(browsing_context_id_list: nil, &event_handler) # &event_handler is optional. It's
just a notation to show that the method accepts a block.

on_browsing_context_destroyed(browsing_context_id_list: nil)

on_navigation_started(browsing_context_id_list: nil)

on_fragment_navigated(browsing_context_id_list: nil)

on_dom_content_loaded(browsing_context_id_list: nil)

on_browsing_context_loaded(browsing_context_id_list: nil)

on_downloaded_started(browsing_context_id_list: nil)

on_navigation_aborted(browsing_context_id_list: nil)

on_navigation_failed(browsing_context_id_list: nil)

on_user_prompt_closed(browsing_context_id_list: nil)
```

Script Module APIs

yet to be implemented

Initializing Script Module

Java

```
ScriptManager scriptManager = new ScriptManager(driver);
ScriptManager scriptManager = new ScriptManager(browserContextId, driver);
ScriptManager scriptManager = new ScriptManager(browserContextIds, driver);
```

C#

```
ScriptManager scriptManager = new ScriptManager(driver);
ScriptManager scriptManager = new ScriptManager(browserContextId, driver);
ScriptManager scriptManager = new ScriptManager(browserContextIds, driver);
```

Javascript

```
scriptManager = new ScriptManager(id, driver)
```

Python

```
manager = ScriptManager(driver);
```

Ruby

```
Script_manager = ScriptManager.new(driver)
```

Command APIs

Java

```
void disownRealmScript(String realmId, List<String> handles)

void disownBrowsingContextScript(String browsingContextId, List<String> handles)
void disownBrowsingContextScript(String browsingContextId, String sandbox, List<String> handles)

ScriptEvaluateResult callFunctionInRealm(String realmId,
                                         String functionDeclaration,
                                         boolean awaitPromise,
                                         Optional<List<ArgumentValue>> argumentValueList,
                                         Optional<ArgumentValue> thisParameter,
                                         Optional<ResultOwnership> resultOwnership)

EvaluateResult callFunctionInBrowsingContext(String browsingContextId,
                                             String functionDeclaration,
                                             boolean awaitPromise,
                                             Optional<List<ArgumentValue>> argumentValueList,
                                             Optional<ArgumentValue> thisParameter,
                                             Optional<ResultOwnership> resultOwnership)
EvaluateResult callFunctionInBrowsingContext(String browsingContextId,
                                             String sandbox,
                                             String functionDeclaration,
                                             boolean awaitPromise,
                                             Optional<List<ArgumentValue>> argumentValueList,
                                             Optional<ArgumentValue> thisParameter,
                                             Optional<ResultOwnership> resultOwnership)
```

```
// OwnernshipModel enum contains value root and none
// Argument can be LocalValue or RemoteReference
// Refer https://w3c.github.io/webdriver-bidi/#command-script-callFunction

EvaluateResult evaluateFunctionInRealm(String realmId, String expression, boolean awaitPromise,
Optional<ResultOwnership> resultOwnership)

EvaluateResult evaluateFunctionInBrowsingContext(String browsingContextId,
                                                 String expression,
                                                 boolean awaitPromise,
                                                 Optional<ResultOwnership> resultOwnership)

EvaluateResult evaluateFunctionInBrowsingContext(String browsingContextId,
                                                 String sandbox,
                                                 String expression,
                                                 boolean awaitPromise,
                                                 Optional<ResultOwnership> resultOwnership)

// OwnernshipModel enum contains value root, none

List<RealmInfo> getRealms(RealmParams params)
List<WindowRealmInfo> getWindowRealms(RealmParams params)
List<DedicatedWorkerRealmInfo> getDedicatedWorkerRealms(RealmParams params)
List<SharedWorkerRealmInfo> getSharedWorkerRealms(RealmParams params)
List<ServiceWorkerRealmInfo> getServiceWorkerRealms(RealmParams params)
List<WorkerRealmInfo> getWorkerRealms(RealmParams params)
List<PaintWorkletRealmInfo> getPaintWorkletRealms(RealmParams params)
List<AudioWorkletRealmInfo> getAudioWorkletRealms(RealmParams params)
List<WorkletRealmInfo> getWorkletRealms(RealmParams params)

//RealmParams will allow passing the realm id, browsing context id or both
```

C#

```
void DisownRealmScript(string realmId, List<string> handles)

void DisownBrowsingContextScript(string browsingContextId, List<string> handles)
void DisownBrowsingContextScript(string browsingContextId, string sandbox, List<string> handles)

ScriptEvaluateResult CallFunctionInRealm(string realmId, CallFunctionParams params )

ScriptEvaluateResult CallFunctionInBrowsingContext(string browsingContextId, CallFunctionParams params)
ScriptEvaluateResult CallFunctionInBrowsingContext(string browsingContextId, string sandbox, CallFunctionParams
params)

// CallFunctionParams class will contain String functionDeclaration, boolean awaitPromise, optional ArgumentValue
list, optional "this" ArgumentValue, optional OwnernshipModel enum
// OwnernshipModel enum contains value root and none
// Argument can be LocalValue or RemoteReference
// Refer https://w3c.github.io/webdriver-bidi/#command-script-callFunction

ScriptEvaluateResult EvaluateFunctionInRealm(string realmId, EvaluateFunctionParams params)

ScriptEvaluateResult EvaluateFunctionInBrowsingContext(string browsingContextId, EvaluateFunctionParams params)
ScriptEvaluateResult EvaluateFunctionInBrowsingContext(string browsingContextId, string sandbox,
EvaluateFunctionParams params)

// CallFunctionParams class will contain String expression, boolean awaitPromise, optional OwnernshipModel enum
// OwnernshipModel enum contains value root, none

List<RealmInfo> GetRealms(RealmParams params)
List<WindowRealmInfo> GetWindowRealms(RealmParams params)
List<DedicatedWorkerRealmInfo> GetDedicatedWorkerRealms(RealmParams params)
List<SharedWorkerRealmInfo> GetSharedWorkerRealms(RealmParams params)
List<ServiceWorkerRealmInfo> GetServiceWorkerRealms(RealmParams params)
List<WorkerRealmInfo> GetWorkerRealms(RealmParams params)
```

```
List<PaintWorkletRealmInfo> GetPaintWorkletRealms(RealmParams params)
List<AudioWorkletRealmInfo> GetAudioWorkletRealms(RealmParams params)
List<WorkletRealmInfo> GetWorkletRealms(RealmParams params)

//RealmParams will allow passing the realm id, browsing context id or both
```

Javascript

```
disownRealmScript(realmId, handles)
disownBrowsingContextScript(browsingContextId, handles, sandbox = null)

callFunctionInRealm(realmId, functionDeclaration, awaitPromise, argumentValueList = null, thisParameter = null,
resultOwnership = null)
callFunctionInBrowsingContext(broesingContextId, functionDeclaration, awaitPromise, argumentValueList = null,
thisParameter = null, resultOwnership = null, sandbox = null)


evaluateInRealm(realmId, expression, awaitPromise, resultOwnership = null)
evaluateInBrowsingContext(browsingContextId, expression, awaitPromise, resultOwnership = null, sandbox = null)

addPreloadScript(functionDeclaration, argumentValueList = [], sandbox = null)
removePreloadScript(script)

getAllRealms()
getRealmsByType(type)
getRealmsInBrowsingContext(browsingContext)
getRealmsInBrowsingContextByType(browsingContext, type)
```

Python

```
disown_realm_script(realmId, handles)
disown_browsing_context_script(browsingContextId, sandbox: None, handles)

call_function_in_realm(realmId, params)
call_function_in_browsing_context(browsingContextId, sandbox: None, params)

// params will be an object of class CallFunctionParams. This class will contain function_declaration,
await_promise, optional argument_value list, optional "this" argument_value, optional ownership_model enum object
// ownership_model enum contains value root and none
// argument_value can be LocalValue or RemoteReference

evaluate_in_realm(realmId, params)
evaluate_in_browsing_context(browsingContextId, sandbox: None, params)

// params will be an object of class EvaluateFunctionParams. This class will contain expression, await_promise,
optional ownership_model
// ownership_model enum contains value root and none

get_realms(params)
get_window_realms(params)
get_dedicated_worker_realms(params)
get_shared_worker_realms(params)
get_service_worker_realms(params)
get_worker_realms(params)
get_paint_worklet_realms(params)
get_audio_worklet_realms(params)
get_worklet_realms(params)

// params will be an object of class RealmParams. This class will allow passing the browsing_context, type of
realm, or both.
```

Ruby

```
disown_realm_script(realm_id, handles)
```

```
disown_browsing_context_script(browsing_context_id, sandbox: nil, handles)

call_function_in_realm(realm_id, params)
call_function_in_browsing_context(browsing_context_id, sandbox: nil, params)

// params will be an object of class CallFunctionParams. This class will contain function_declaration,
await_promise, optional argument_value list, optional "this" argument_value, optional ownership_model enum object
// ownership_model enum contains value root and none
// argument_value can be LocalValue or RemoteReference

evaluate_in_realm(realm_id, params)
evaluate_in_browsing_context(browsing_context_id, sandbox: nil, params)

// params will be an object of class EvaluateFunctionParams. This class will contain expression, await_promise,
optional ownership_model
// ownership_model enum contains value root and none

get_realms(params)
get_window_realms(params)
get_dedicated_worker_realms(params)
get_shared_worker_realms(params)
get_service_worker_realms(params)
get_worker_realms(params)
get_paint_worklet_realms(params)
get_audio_worklet_realms(params)
get_worklet_realms(params)

// params will be an object of class RealmParams. This class will allow passing the browsing_context, type of
realm, or both.
```

Event APIs

Java

```
void onRealmCreated(Consumer<RealmInfo> consumer)

void onWindowRealmCreated(Consumer<WindowRealmInfo> consumer)

void onDedicatedWorkerRealmCreated(Consumer<DedicatedWorkerRealmInfo> consumer)

void onSharedWorkerRealmCreated(Consumer<SharedWorkerRealmInfo> consumer)

void onServiceWorkerRealmCreated(Consumer<ServiceWorkerRealmInfo> consumer)

void onWorkerRealmRealmCreated(Consumer<WorkerRealmInfo> consumer)

void onPaintWorkletRealmCreated(Consumer<PaintWorkletRealmInfo> consumer)

void onAudioWorkletRealmCreated(Consumer<AudioWorkletRealmInfo> consumer)

void onWorkletRealmCreated(Consumer<WorkletRealmInfo> consumer)

void onRealmDestroyed(Consumer<String> consumer)
```

C#

```
void OnRealmCreated(EventHandler<RealmInfoArgs> handler)

void OnWindowRealmCreated(EventHandler<WindowRealmInfoArgs> handler)

void OnDedicatedWorkerRealmCreated(EventHandler<DedicatedWorkerRealmInfoArgs> consumer)

void OnSharedWorkerRealmCreated(EventHandler<SharedWorkerRealmInfoArgs> handler)
```

```
void OnServiceWorkerRealmCreated(EventHandler<ServiceWorkerRealmInfoArgs> handler)

void OnWorkerRealmRealmCreated(EventHandler<WorkerRealmInfoArgs> handler)

void OnPaintWorkletRealmCreated(EventHandler<PaintWorkletRealmInfoArgs> handler)

void OnAudioWorkletRealmCreated(EventHandler<AudioWorkletRealmInfoArgs> handler)

void OnWorkletRealmCreated(EventHandler<WorkletRealmInfoArgs> handler)

void OnRealmDestroyed(EventHandler<RealmArgs> handler)
```

Javascript

// yet to be implemented

```
onRealmCreated(browsingContextIds = null, handler)

onWindowRealmCreated(browsingContextIds = null, handler)

onDedicatedWorkerRealmCreated(browsingContextIds = null, handler)

onSharedWorkerRealmCreated(browsingContextIds = null, handler)

onServiceWorkerRealmCreated(browsingContextIds = null, handler)

onWorkerRealmRealmCreated(browsingContextIds = null, handler)

onPaintWorkletRealmCreated(browsingContextIds = null, handler)

onAudioWorkletRealmCreated(browsingContextIds = null, handler)

onWorkletRealmCreated(browsingContextIds = null, handler)

onRealmDestroyed(browsingContextIds = null, handler)
```

Python

```
on_realm_created(browsingContextIds = None, eventHandler)

on_window_realm_created(browsingContextIds = None, eventHandler)

on_dedicated_worker_realm_created(browsingContextIds = None, eventHandler)

on_shared_worker_realm_created(browsingContextIds = None, eventHandler)

on_service_worker_realm_created(browsingContextIds = None, eventHandler)

on_worker_realm_created(browsingContextIds = None, eventHandler)

on_paint_worklet_realm_created(browsingContextIds = None, eventHandler)

on_audio_worklet_realm_created(browsingContextIds = None, eventHandler)

on_worklet_realm_created(browsingContextIds = None, eventHandler)

on_realm_destroyed(browsingContextIds = None, eventHandler)
```

Ruby

```
on_realm_created(browsing_context_ids: nil, &event_handler)  // &event_handler is optional. It's just a notation
to show that the method accepts a block.

on_window_realm_created(browsing_context_ids: nil, &event_handler)
```

```
on_dedicated_worker_realm_created(browsing_context_ids: nil, &event_handler)

on_shared_worker_realm_created(browsing_context_ids: nil, &event_handler)

on_service_worker_realm_created(browsing_context_ids: nil, &event_handler)

on_worker_realm_created(browsing_context_ids: nil, &event_handler)

on_paint_worklet_realm_created(browsing_context_ids: nil, &event_handler)

on_audio_worklet_realm_created(browsing_context_ids: nil, &event_handler)

on_worklet_realm_created(browsing_context_ids: nil, &event_handler)

on_realm_destroyed(browsing_context_ids: nil, &event_handler)
```

Network Module APIs

Initializing Network Module

Javascript

```
const inspector = await NetworkInspector(driver)
```

Ruby

```
inspector = NetworkInspector.new(driver)
```

Event APIs

Javascript

```
async beforeRequestSent(callback)
async responseStarted(callback)
async responseCompleted(callback)
```

Ruby

```
def before_request_sent(&block)
def response_started(&block)
def response_completed(&block)
```

Unified BiDi APIs usage example

Java:

```
FirefoxOptions options = new FirefoxOptions();
options.setCapability("webSocketUrl", true);

Webdriver driver = new FirefoxDriver(options);

LogInspector logInspector = new LogInspector(driver);
logInspector.onConsoleEntry(logEntry -> {
   System.out.println(logEntry.getText());
   System.out.println(logEntry.getStackTrace());
});

driver.get("http://selenium.dev");
```

```
driver.quit();
```

C#:

```
var options = new FirefoxOptions();
options.AddAdditionalOption("webSocketUrl", true);

var driver = new FirefoxDriver();
LogInspector logInspector = new LogInspector(driver);

EventHandler<ConsoleLogEntryArgs> handler = (sender, eventArgs) =>
   {
       Console.WriteLine(eventArgs.Text);
       Console.WriteLine(eventArgs.StackTrace);

   };

logInspector.OnConsoleLog(handler);
// internally it will subscribe to the event for C#, ex: this.ConsoleLogEventCalled +=handler

driver.Navigate().GoToUrl("https://selenium.dev");

driver.Quit();
```

Javascript:

```
const driver = new Builder()
    .forBrowser('firefox')
    .setFirefoxOptions(new firefox.Options().enableBidi())
    .build();

logInspector = new LogInspector(driver)

await logInspector.onConsoleEntry(function (logEntry) {
   console.log(logEntry.text)
   console.log(logEntry.stackTrace)
}

await driver.get("https://selenium.dev");
await driver.quit();
```

Python:
// yet to be implemented

```
def handler(log_entry):
  print(log_entry.get_text())
  print(log_entry.get_stacktrace())


firefox_options = webdriver.FirefoxOptions()
firefox_options.set_capability("webSocketUrl", "true")

driver = webdriver.Firefox(firefox_options)

logInspector = LogInspector(driver);

driver.on_console_log(handler)

driver.get("http://www.google.com")

driver.quit()
```

Ruby

```
options = Selenium::WebDriver::Options.firefox
options.add_option(:web_socket_url, true)
driver = Selenium::WebDriver.for :firefox, options: options

log_inspector = LogInspector.new(driver)
log_inspector.on_console_entry {|log_entry|
      puts "#{log_entry.text}"
      puts "#{log_entry.stack_trace}"
}
driver.get 'https://selenium.dev'
driver.quit
```

Use-cases

So far the APIs defined try to map the commands and events defined in https://w3c.github.io/webdriver-bidi/. While users can use this directly as needed, there are popular uses-cases which use a combination of the BiDi or CDP APIs to address testing requirements.

Note: This section aims to map the existing use-cases defined in https://www.selenium.dev/documentation/webdriver/bidirectional/bidi_api/. It will be further extended as more use-cases come in. (Context: https://www.w3.org/2022/08/24-webdriver-minutes.html)

Register Basic Auth

Requires APIs similar to https://chromedevtools.github.io/devtools-protocol/tot/Fetch/
Related BiDi issue https://github.com/w3c/webdriver-bidi/issues/66

Mutation Observation

Current implementation in Selenium uses a javascript scrip and requires an event similar to
https://chromedevtools.github.io/devtools-protocol/tot/Runtime/#event-bindingCalled

Listen to console.log events

Refer example and Log Module Event APIs

Listen to JS Exceptions

Refer Log Module Event APIs

Usage example in Java

```
FirefoxOptions options = new FirefoxOptions();
options.setCapability("webSocketUrl", true);

Webdriver driver = new FirefoxDriver(options);

LogInspector logInspector = new LogInspector(driver);
logInspector.onJavaScriptException(logEntry -> {
   System.out.println(logEntry.getText());
   System.out.println(logEntry.getStackTrace());
});

driver.get("http://selenium.dev");

driver.quit();
```

Usage example in Javascript

```
const driver = new Builder()
    .forBrowser('firefox')
    .setFirefoxOptions(new firefox.Options().enableBidi())
    .build();

logInspector = new LogInspector(driver)
```

```
await logInspector.onJavascriptException(function (logEntry) {
   console.log(logEntry.text)
   console.log(logEntry.stackTrace)
}

await driver.get("https://selenium.dev");
await driver.quit();
```

Usage example in ruby

```
options = Selenium::WebDriver::Options.firefox
options.add_option(:web_socket_url, true)
driver = Selenium::WebDriver.for :firefox, options: options

log_inspector = LogInspector.new(driver)
log_inspector.on_javascript_exception {|log_entry|
     puts "#{log_entry.text}"
     puts "#{log_entry.stack_trace}"
}
driver.get 'https://selenium.dev'
driver.quit
```

Network Interception

Requires APIs similar to https://chromedevtools.github.io/devtools-protocol/tot/Fetch/
Related BiDi issue https://github.com/w3c/webdriver-bidi/issues/66