# Writing Browser Extensions in Kotlin

Kirill Rakhman (@Cypressious)
busradar.com

## KotlinJS

- No longer expiremental since 1.1.0
- Can run anywhere where JS runs
  - Websites
  - NodeJS
  - Browser Extensions
- Can call JS
  - the *dynamic* way
  - the *static* way
- Benefits from all IDE features you know from Kotlin on the JVM
- Question: Does it make writing browser extensions more pleasant?

## About `dynamic`

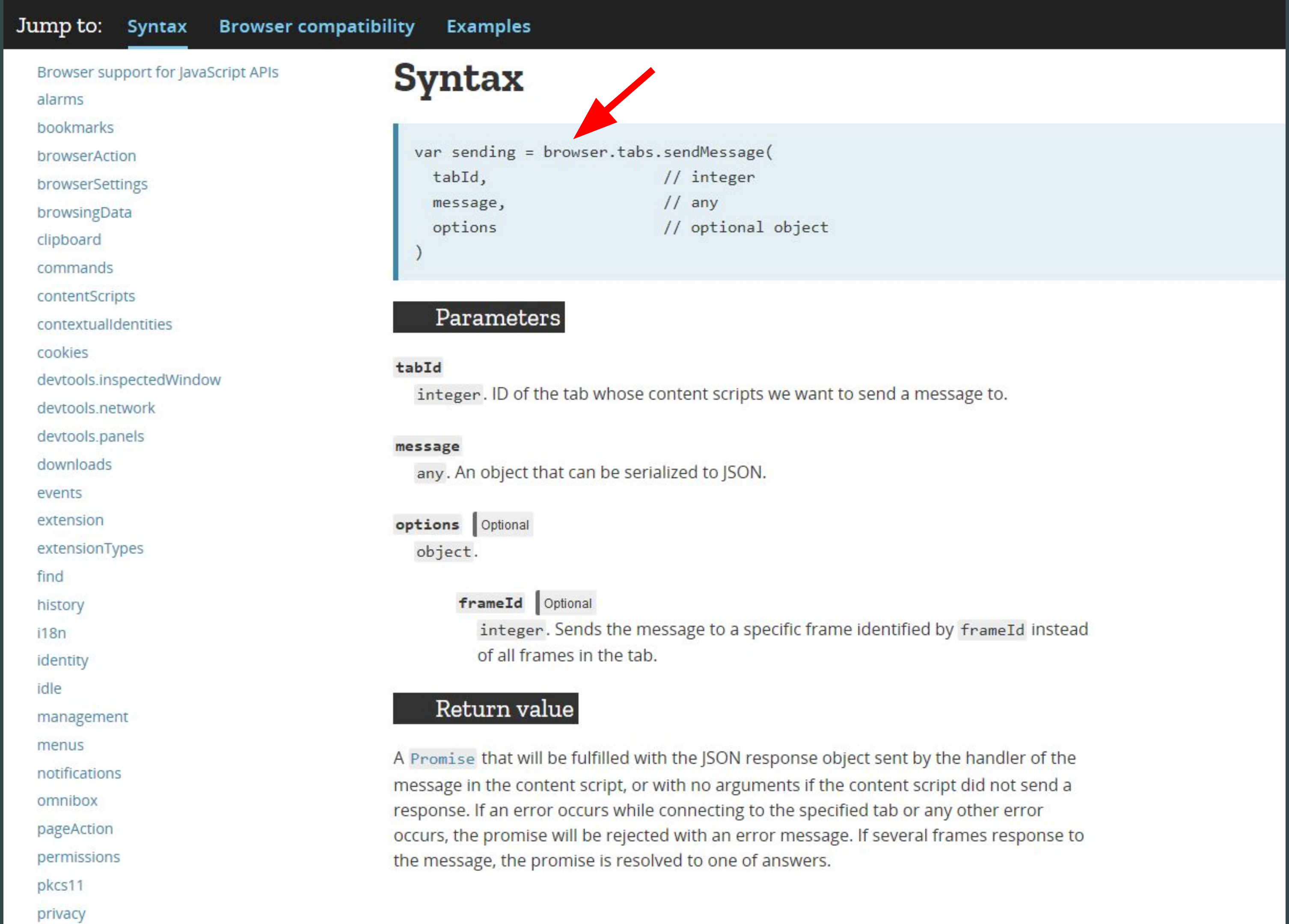
```
val a: Any = js("{}")
a.foo()

val b: dynamic = a.asDynamic()
b.foo()
b["baz"] = "qux"
```

## Calling external code

```
external val foo: dynamic

external class External {
    fun callMe()
}

fun main() {
    foo.bar()
    foo.baz.qux = false

    val e = External()
    e.callMe()
    e.somethingElse()
}
```

## WebExtensions

- A new cross-browser extension API standard
- Implemented by Firefox, Chrome, Opera and Edge to varying degrees
- We’re going to focus on Firefox



## Syntax

```
var sending = browser.tabs.sendMessage(
  tabId,                    // integer
  message,                  // any
  options                   // optional object
)
```

### Parameters

**`tabId`**
`integer`. ID of the tab whose content scripts we want to send a message to.

**`message`**
`any`. An object that can be serialized to JSON.

**`options`** Optional
`object`.

**`frameId`** Optional
`integer`. Sends the message to a specific frame identified by `frameId` instead of all frames in the tab.

### Return value

A `Promise` that will be fulfilled with the JSON response object sent by the handler of the message in the content script, or with no arguments if the content script did not send a response. If an error occurs while connecting to the specified tab or any other error occurs, the promise will be rejected with an error message. If several frames response to the message, the promise is resolved to one of answers.

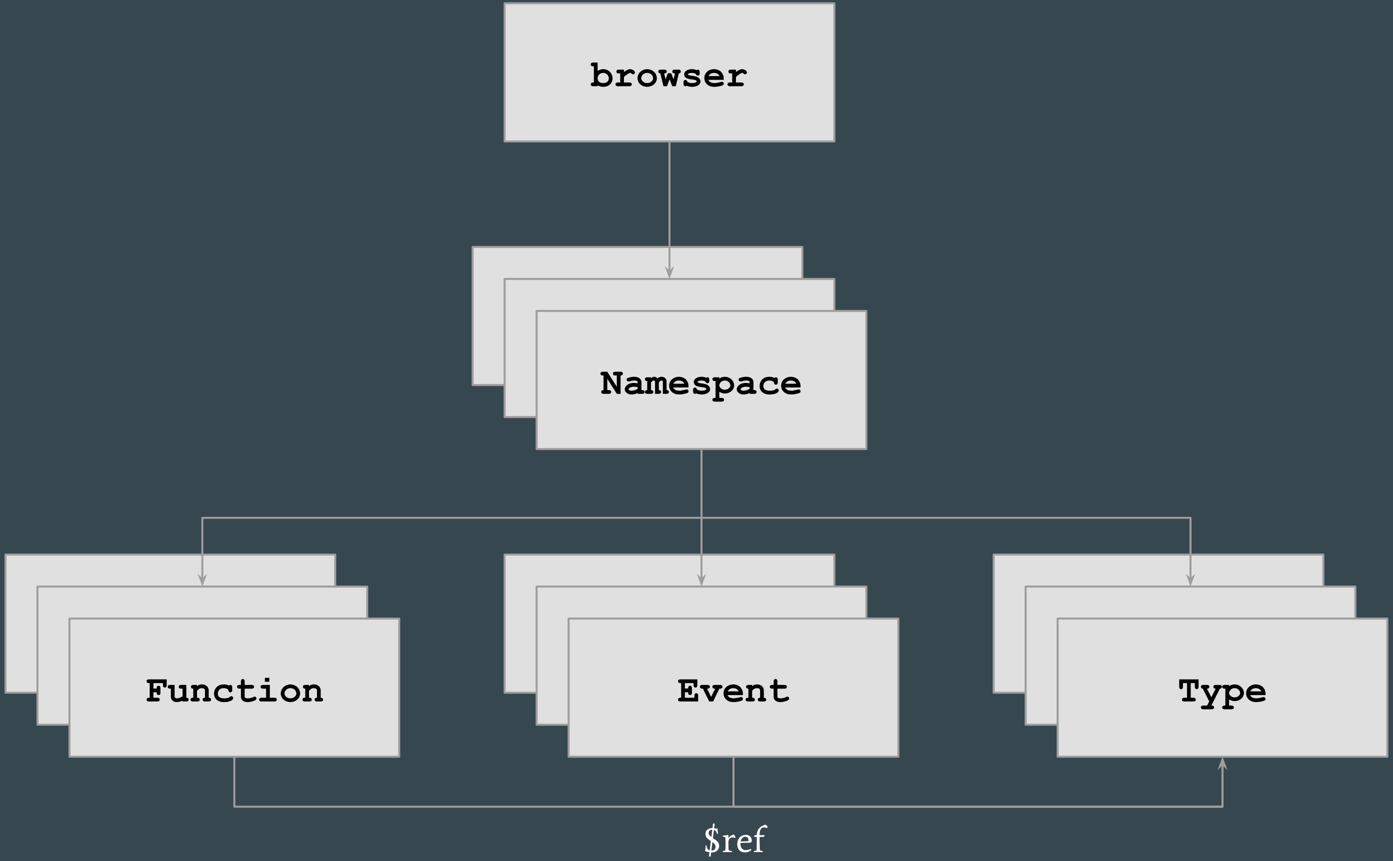
## Calling the WebExtensions API from Kotlin

- `external val browser: dynamic`
  - easily implemented
  - no compile-time safety
  - no IDE support
- writing external declarations
  - extra effort
  - compile-time safety (assuming you wrote the declarations correctly)
  - IDE support
- code generation from schema
  - no extra effort (because I did it ;)
  - compile-time safety guaranteed by official schema
  - awesome IDE support, including autocompletion, KDoc, deprecation, …

## The schema

```
{
 "name": "sendMessage",
 "type": "function",
 "description": "...",
 "async": "responseCallback",
 "parameters": [
   ...,
   {
     "type": "object",
     "name": "options",
     "properties": {
       "frameId": {
         "type": "integer",
         "optional": true,
         "minimum": 0,
       }
     },
     "optional": true
   }
 ]
}
```

# API Structure



## Transforming JS objects

```
"parameters": [
 {
   "type": "object",
   "name": "createProperties",
   "properties": {
     "windowId": {
       "type": "integer",
       "optional": true
     },
     "url": {
       "type": "string",
       "optional": true
     },
     "active": {
       "type": "boolean",
       "optional": true
     }
 }
]
```

```
class CreateProperties(
        var windowId: Int? = null,
        var url: String? = null,
        var active: Boolean? = null
)
```

# Transforming JS objects (alternative)

```
"parameters": [
 {
   "type": "object",
   "name": "createProperties",
   "properties": {
     "windowId": {
       "type": "integer",
       "optional": true
     },
     "url": {
       "type": "string",
       "optional": true
     },
     "active": {
       "type": "boolean",
       "optional": true
     }
 }
]
```

```kotlin
external interface CreateProperties {
        var windowId: Int?
        var url: String?
        var active: Boolean?
}

fun CreateProperties(
   windowId: Int? = null,
   url: String? = null,
   active: Boolean? = null
) : CreateProperties {

   val o: dynamic = js("{}")
   o.windowId = windowId
   o.url = url
   o.active = active

   return o
}
```

https://youtrack.jetbrains.com/issue/KT-21653

## Transforming Map-like objects

```
"formData": {
 "type": "object",
 "properties": {},
 "additionalProperties": {
   "type": "array",
   "items": { "type": "string" }
 }
}



{
 "type": "object",
 "patternProperties": {
   "^[1-9]\\d*$": {"$ref": "ImageDataType"}
 }
}
```

```
class FormData {

   inline operator fun get(key: String):
Array<String> = asDynamic()[key]

   inline operator fun set(key: String,
value: Array<String>) {
       asDynamic()[key] = value
   }

}
```

## Transforming union types in function parameters

```json
"functions": [
 {
   "name": "get",
   "type": "function",
   "async": "callback",
   "parameters": [
     {
       "name": "idOrIdList",
       "choices": [
         {
           "type": "string"
         },
         {
           "type": "array",
           "items": {
             "type": "string"
           }
         }
       ]
     }
 ...
```

```kotlin
external class BookmarksNamespace {

   fun get(idOrIdList: String):
Promise<Array<BookmarkTreeNode>>

   fun get(idOrIdList: Array<String>):
Promise<Array<BookmarkTreeNode>>

}
```

## Transforming union types

```
{
 "id": "OnClickData",
 "type": "object",
 "properties": {
   "menuItemId": {
     "choices": [
       { "type": "integer" },
       { "type": "string" }
     ]
   },
 ...
```

```
class OnClickData(
   var menuItemId: MenuItemId
)

typealias MenuItemId = Any
```

:(

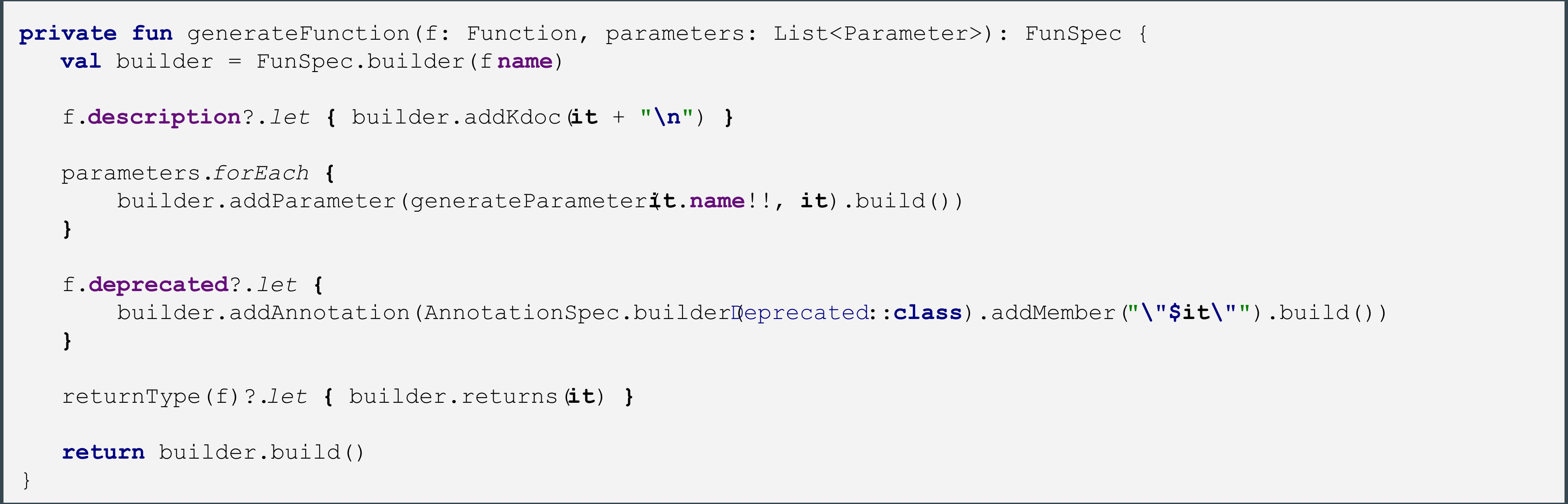
## Transforming Events

```
"events": [
 {
   "name": "onCreated",
   "type": "function",
   "parameters": [
     {
       "$ref": "Tab",
       "name": "tab"
     }
   ]
 },
 ...
```

```
external class Event<in T> {
   fun addListener(listener:  T)

   fun removeListener(listener:  T)

   fun hasListener(listener:  T): Boolean
}


val onCreated: Event<(tab: Tab) -> Unit>
```

# Generating Code

- KotlinPoet is “a Kotlin and Java API for generating .kt source files.”
- Heavy use of the Builder pattern
- Make use of information like optional parameters, deprecation, KDoc, ...



```kotlin
private fun generateFunction(f: Function, parameters: List<Parameter>): FunSpec {
   val builder = FunSpec.builder(f name)

   f.description?.let { builder.addKdoc(it + "\n") }

   parameters.forEach {
       builder.addParameter(generateParameter(it.name!!, it).build())
   }

   f.deprecated?.let {
       builder.addAnnotation(AnnotationSpec.builder(Deprecated::class).addMember("\"$it\"").build())
   }

   returnType(f)?.let { builder.returns(it) }

   return builder.build()
}
```

## The generated code

```kotlin
external val browser: Browser

external class Browser {
    val tabs: TabsNamespace
    ...
}

external class TabsNamespace {
    /** ... */
    fun move(tabIds: Int, moveProperties: MoveProperties): Promise<Tabs2>

    /** ... */
    fun move(tabIds: Array<Int>, moveProperties: MoveProperties): Promise<Tabs2>

    ...
}

class MoveProperties(...)
```

https://github.com/cypressious/kotlin-webextensions-declarations

## Code Demo

## Future improvements

- Tooling
    - Multiple projects for different output files are annoying
- Keeping up with API updates
    - Auto generate using Gradle plugin?
- Use external interfaces instead of classes
    - Smaller compiled JS
- Make it compatible with Chrome
    - Is not Promise based
    - Base object
    - Polyfill[1] already exist

[1] https://github.com/mozilla/webextension-polyfill

## Thank you for your attention

Kirill Rakhman

cypressious

rakhman.info

## Writing the plugin

```
buildscript {
   ext.kotlin_version = '1.2.70'

   repositories {
       mavenCentral()

   }
   dependencies {
       classpath
"org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
   }
}

version '0.1'
```

```
allprojects {
   apply plugin: 'kotlin2js'
   apply plugin: 'kotlin-dce-js'

   repositories {
       mavenCentral()
       maven { url 'https://jitpack.io' }
   }

   dependencies {
       compile
"org.jetbrains.kotlin:kotlin-stdlib-js:$kotlin_version"
       compile
'com.github.cypressious.kotlin-webextensions-declarations:w
ebextensions-declarations:v0.1'
   }

   compileKotlin2Js {
       kotlinOptions.sourceMap = true
       kotlinOptions.sourceMapEmbedSources = "always"
   }
}
```

https://github.com/cypressious/webextension-search-kotlin-docs

## Writing the plugin

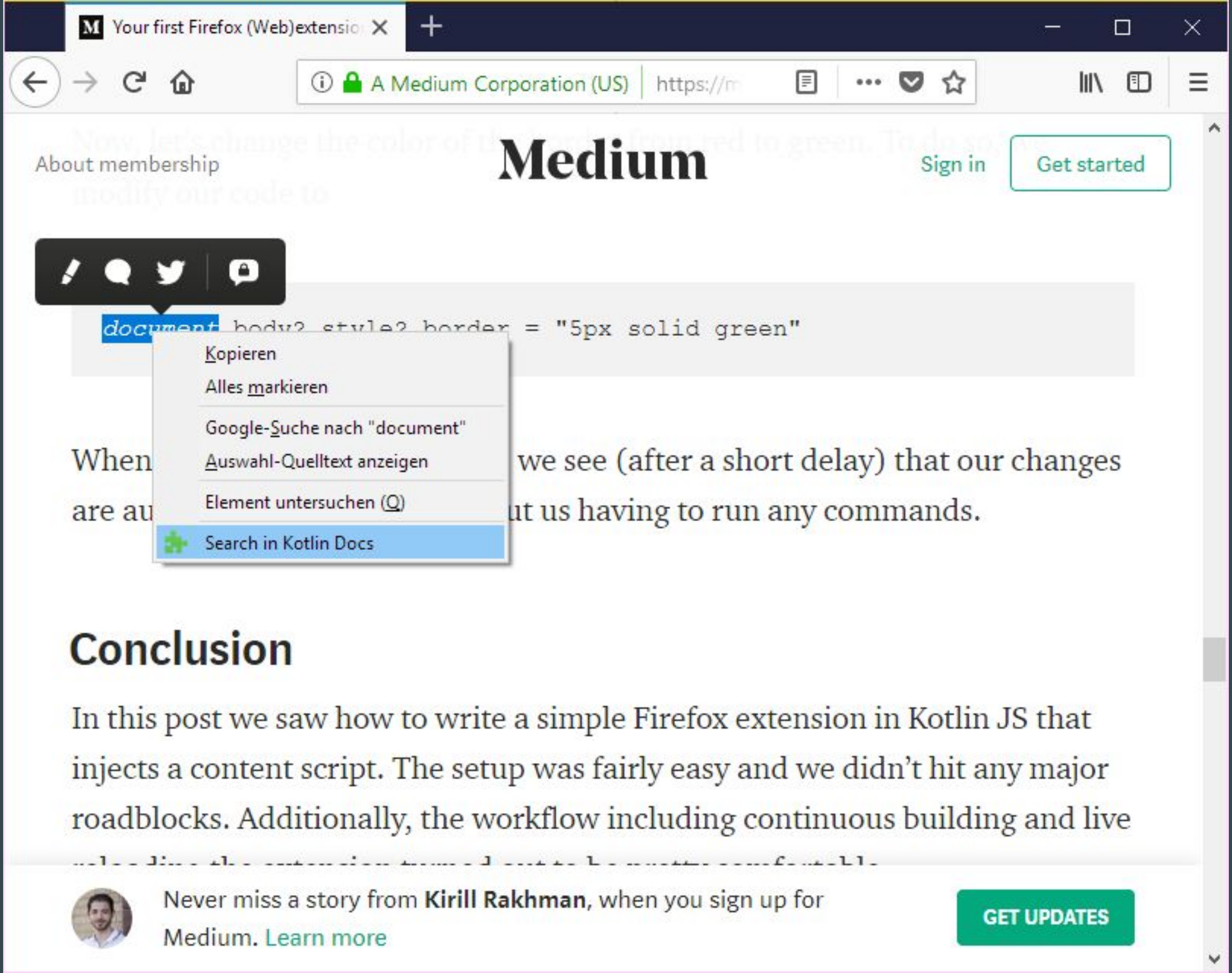
```json
{
 "description": "Adds a context menu item to search for the selected word in the Kotlin documentation",
 "manifest_version": 2,
 "name": "Search Kotlin",
 "version": "1.0",
 "icons": {
 },
 "background": {
   "scripts": [
     "build/kotlin-js-min/main/kotlin.js",
     "build/kotlin-js-min/main/declarations.js",
     "build/kotlin-js-min/main/ff-search-kotlin.js"
   ]
 },
 "permissions": [
   "menus"
 ]
}
```

https://github.com/cypressious/webextension-search-kotlin-docs

# Writing the plugin

```kotlin
import menus.CreateProperties
import webextensions.browser

fun main(args: Array<String>) {
   browser.menus.create(CreateProperties(
           id = "search-kotlin",
           title = "Search in Kotlin Docs",
           contexts = arrayOf("selection")
   ))

   browser.menus.onClicked.addListener { info, tab ->
       when (info.menuItemId) {
           "search-kotlin" -> {
               browser.tabs.create(tabs.CreateProperties(
                       url = "http://kotlinlang.org/?q=${info.selectionText}&p=0"
               ))
           }
       }
   }
}
```

https://github.com/cypressious/webextension-search-kotlin-docs



Your first Firefox (Web)extensio

A Medium Corporation (US) https://m

About membership

Medium

Sign in

Get started

```
document.body?.style?.border = "5px solid green"
```

Kopieren
Alles markieren
Google-Suche nach "document"
Auswahl-Quelltext anzeigen
Element untersuchen (Q)
Search in Kotlin Docs

When … we see (after a short delay) that our changes are au… …ut us having to run any commands.

## Conclusion

In this post we saw how to write a simple Firefox extension in Kotlin JS that injects a content script. The setup was fairly easy and we didn't hit any major roadblocks. Additionally, the workflow including continuous building and live

Never miss a story from **Kirill Rakhman**, when you sign up for Medium. Learn more

GET UPDATES