Globalne this

Podsumowując omówione do tej pory środowiska JS, program może lub też nie:

- Zadeklarować zmienną globalną w zakresie najwyższego poziomu przy użyciu deklaracji `var` lub `function` – bądź `let`, `const` i `class`.
- Dodać również deklaracje zmiennych globalnych jako właściwości obiektu zakresu globalnego, jeśli w deklaracji użyto `var` lub `function`.
- Odwołać się do obiektu zakresu globalnego (w celu dodania lub pobrania zmiennych globalnych jako właściwości) przy użyciu `window`, `self` lub `global`.

Moim zdaniem, wielu programistów nie docenia złożoności problemu dostępu i działania zakresu globalnego, co zostało zilustrowane w dotychczasowej części rozdziału. Jednak ta złożoność staje się najbardziej oczywista, gdy próbujemy ustalić uniwersalne odwołanie do obiektu zakresu globalnego.

Kolejna „sztuczka" umożliwiająca uzyskanie odwołania do obiektu zakresu globalnego wygląda następująco:

```
const theGlobalScopeObject =
    (new Function("return this"))();
```

Uwaga

Na podstawie kodu przechowywanego w wartości łańcucha można dynamicznie skonstruować funkcję, korzystając z konstruktora `Function()`, podobnie do `eval(..)` (patrz „Oszukiwanie: zmienianie zakresu w czasie wykonania" w rozdziale 1). Taka funkcja zostanie automatycznie uruchomiona w trybie nieścisłym (ze względu na starsze wersje), gdy wywołamy ją przy użyciu normalnego wywołania funkcji `()`, jak w tym przykładzie. Jej `this` będzie wskazywać obiekt globalny. Dodatkowe informacje o określaniu wiązań `this` znaleźć można w trzeciej książce z serii zatytułowanej *Objects & Classes*[1].

A zatem mamy do dyspozycji `window`, `self`, `global` i brzydką sztuczkę `new Function(..)`. To wiele różnych metod uzyskiwania dostępu do obiektu zakresu globalnego. Każda z nich ma pewne wady i zalety.

Czemu nie wprowadzić jeszcze jednej metody!?

[1] Książka w trakcie opracowywania (przyp. red.).

W wersji ES2020 wreszcie zdefiniowano ustandaryzowane odwołanie do obiektu zakresu globalnego o nazwie `globalThis`. Dlatego w zależności od wersji silnika JS, w jakim uruchamiany jest kod, można użyć `globalThis` zamiast dowolnej z tych metod.

Moglibyśmy nawet spróbować zdefiniować polyfill dla różnych środowisk, który jest bezpieczniejszy w różnych środowiskach JS poprzedzających `globalThis`, taki jak:

```
const theGlobalScopeObject =
      (typeof globalThis != "undefined") ? globalThis :
      (typeof global != "undefined") ? global :
      (typeof window != "undefined") ? window :
      (typeof self != "undefined") ? self :
      (new Function("return this"))();
```

Uff! Z pewnością nie jest to idealne rozwiązanie, ale może się przydać, gdy będziesz potrzebować rzetelnego odwołania do zakresu globalnego.

Tak na marginesie, proponowana nazwa `globalThis` wzbudziła pewne kontrowersje, gdy funkcja była dodawana do JS. A dokładniej, ja i wiele innych osób uważaliśmy, że użycie „this" w jej nazwie jest mylące, ponieważ odwołanie do tego obiektu miało służyć do uzyskiwania dostępu do zakresu globalnego, a nie jakiegoś globalnego/domyślnego wiązania `this`. Pod uwagę brano wiele innych nazw, ale z różnych przyczyn zostały one wykluczone. Niestety wybrana została nazwa, która była traktowana jako ostateczność. Jeśli planujesz wykorzystywać w swoich programach obiekt zakresu globalnego, zdecydowanie zalecam wybranie lepszej nazwy, takiej jak (groteskowo długa, ale precyzyjna!) `theGlobalScopeObject` użyta w tym przykładzie.

Zrozumienie zakresu globalnego

Zakres globalny jest obecny i ważny w każdym programie JS, mimo iż nowoczesne wzorce organizacji kodu w moduły ograniczają konieczność przechowywania identyfikatorów w tej przestrzeni nazw.

Mimo to, w miarę jak nasz kod wychodzi coraz bardziej poza granice przeglądarki, szczególnie ważne jest dobre zrozumienie różnic w działaniu zakresu globalnego (i obiektu zakresu globalnego!) w różnych środowiskach JS.

Skoro mamy już wyraźniejszy ogólny obraz zakresu globalnego, w kolejnym rozdziale ponownie zagłębimy się w szczegóły zakresu leksykalnego, analizując, jak i kiedy można korzystać ze zmiennych.