

# Console output support for Worklets

Author: Hiroki Nakagawa ([nhiroki@chromium.org](mailto:nhiroki@chromium.org))

Last update: October 24, 2016

Issue: <https://crbug.com/646559>

Status: DONE

Visibility: **Public**

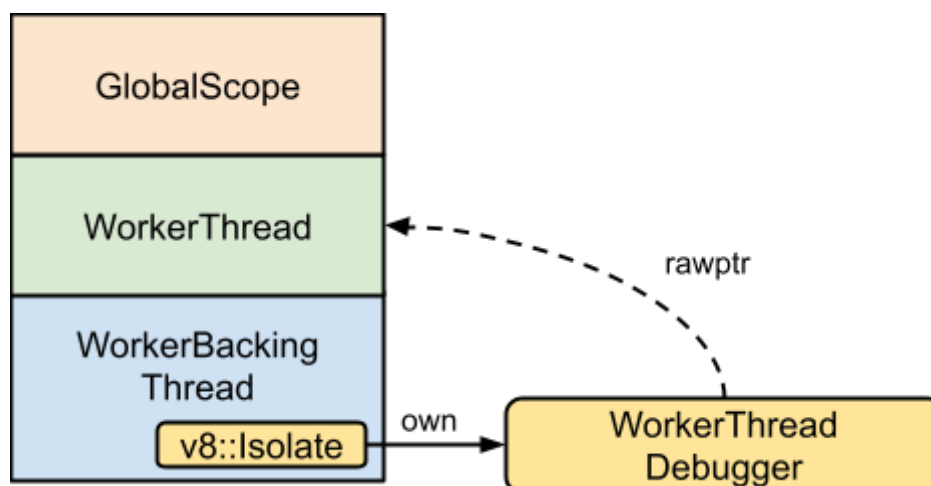
## Objective

Console output is now disabled on Worklets because worker's debugger infra doesn't support multiple global scopes on a single thread. This document explains a problem of the current architecture and proposes a solution.

Note that this addresses only console output support for Worklets. Other features of DevTools like setting break points are out of scope.

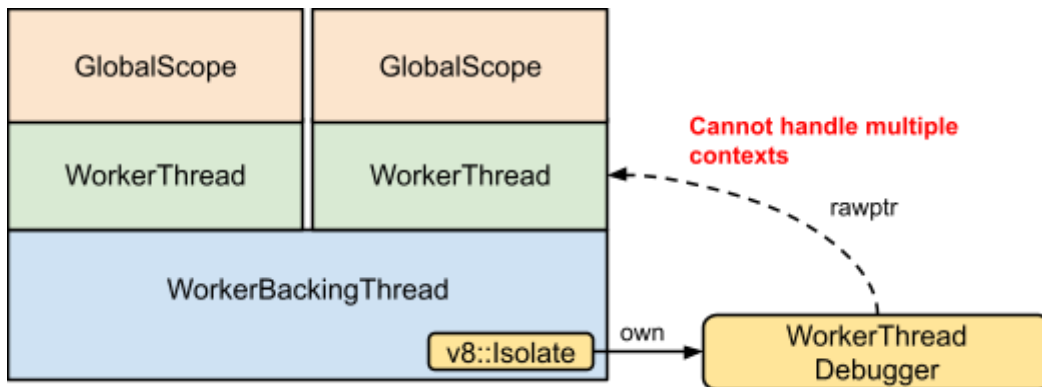
## Problem

WorkerThreadDebugger is an implementation of V8's debugger client. The debugger was designed for inspecting one global scope on one WorkerThread. This is also used for console output. Figure 1 describes this architecture. WorkerBackingThread is a key class here. It hosts `v8::Isolate`, which is in turn associated with WorkerThreadDebugger. WorkerThread exclusively hosts just one GlobalScope that inherits ExecutionContext.



**Figure 1) Current architecture of WorkerThreadDebugger on Workers**

This architecture had worked well until Worklets emerged. Worklets have somewhat different architecture from traditional workers: multiple WorkerThreads can share a single WorkerBackingThread (`v8::Isolate`) as Figure 2.

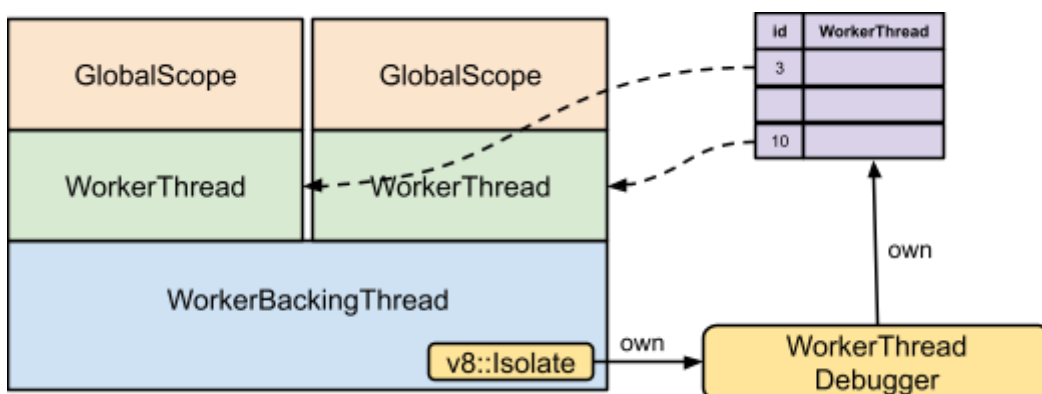


**Figure 2) Current architecture of WorkerThreadDebugger on Worklets (not work)**

This difference makes WorkerThreadDebugger confused because the debugger cannot handle multiple global scopes on one `v8::Isolate`. This causes a renderer crash when multiple Worklets run on the same `v8::Isolate`, so currently the debugger is disabled (is not attached with the isolate) on Worklets (see `WorkerThread::shouldAttachThreadDebugger()`)

## Solution

To support console output for Worklets, we'll change WorkerThreadDebugger to manage ID-WorkerThread map so that the debugegr can handle multiple global scopes (see Figure 3). WorkerThreadDebugger looks up an ID provided by a caller in the map, and routes operation requests to an appropriate context.



**Figure 3) Proposed architecture of WorkerThreadDebugger on Worklets**

Actually, this ID mechanism has already been in ThreadDebugger interface. The interface defines the ID as `ContextGroupId`. `MainThreadDebugger` uses `ContextGroupId` for supporting multiple contexts (frames). On the other hand, `WorkerThreadDebugger` has used a fixed ID for the context group ID and usually just ignores it. We need to replace it with a unique ID among execution contexts on a `WorkerBackingThread`. Maybe a unique ID should be assigned per `WorkerThread` or `GlobalScope`.

This mechanism also needs to change lifetime of `WorkerThreadDebugger`. In the current implementation, `WorkerThreadDebugger` is created on `WorkerThread` initialization. Instead, we need to create it on `WorkerBackingThread` initialization and add/remove `WorkerThread` to/from the map when `WorkerThread` is initialized/destroyed.

At this point, `WorkerThreadDebugger` would be ready to work on Worklets, so we enable it by flipping `WorkerThread::shouldAttachThreadDebugger()` for Worklets.