ecma INTERNATIONAL

Technical Report

ECMA TR/114

1st Edition / June 2026

Runtime keys

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

is the registered trademark of Ecma International

Contents

Introduction

This Technical Report defines a canonical source for identifiers of unique ECMAScript runtime environments, known as *runtime keys*.

This Ecma Technical Report was developed by Technical Committee 55 and was adopted by the General Assembly of June 2026

ALTERNATIVE COPYRIGHT NOTICE AND COPYRIGHT LICENSE

Runtime keys

1 Scope

Runtime keys provide a consistent and predictable mechanism for identifying ECMAScript runtimes in a variety of contexts, including but not limited to project configuration files, package manifests, conditional exports, and runtime detection mechanisms. This document focuses on providing informative identifiers primarily for web servers, though other ECMAScript server environments may find it useful. These keys are not intended to be used in web browsers, which should be referenced by other mechanisms such as the browserslist project.

This Report defines:

- the canonical source of runtime keys and associated metadata,
- guidance for future ECMAScript runtimes to submit runtime keys, and
- TC55's governance process for adding and modifying runtime keys to the Report.

NOTE 1 This Report does not define *how* runtime keys should be used by tools or libraries. The purpose of this Report is to prevent conflicts and provide a reliable, authoritative source of runtime identifiers.

NOTE 2 Inclusion in this Report does not imply that the specified runtime is fully conformant with any Ecma specification, including the WinterTC Minimum Common API. Inclusion does not imply endorsement of any kind.

2 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

2.1
ECMAScript runtime
implementation of the ECMA-262 ECMAScript language specification

2.2
runtime key
unique string identifier that represents a specific ECMAScript runtime environment

3 Runtime key structure

3.1 Key format

A runtime key satisfies all the following:

- It is a string value.
- It is usable in common configuration formats such as JSON and YAML.
- It meaningfully represents the associated runtime environment.
- It is simple and unambiguous.

NOTE Future versions of this Report may define a formal grammar for allowable characters in runtime keys.

3.2 Key uniqueness

A runtime key included in this Report will not conflict with the following:

- any other existing runtime key in this Report, or
- any existing Browserslist *browser* <https://github.com/browserslist/browserslist?tab=readme-ov-file#browsers> entries.

> NOTE This explicitly excludes references to *server* runtimes within Browserslist.

3.3 Key naming guidelines

Runtime keys are subject to expert review by TC55. Acceptable keys will:

- avoid being too similar to existing keys,
- avoid similarity to common English words or offensive terms,
- avoid being too generic or similar to general terminology such as "Web Runtimes", "Edge Runtimes", or "JavaScript Runtimes", and
- demonstrate reasonable specificity and clarity.

3.4 Runtime existence validation

The runtime environment represented by a key must actually exist, whether as open source, source-available, or proprietary software. Entries in this Report exist to identify real runtimes, not to reserve names for future projects.

> NOTE A runtime should demonstrate "proof of use" before being added to the Report. Examples include usage in package.json exports fields, published documentation, or active community adoption.

3.5 Runtime metadata

Each runtime key entry includes the following metadata:

- **Organization** (required): The organization or individual responsible for the runtime
- **Name** (required): The human-readable name of the runtime
- **Key** (required): The unique string identifier
- **Description** (required): A brief description of the runtime
- **Website** (optional): The primary website for the runtime
- **Repository** (optional): The source code repository URL, if available
- **Deprecated** (required): A boolean indicating whether the key is deprecated

4 Canonical source for runtime keys

The following table lists runtime keys submitted to TC55 and reviewed by the Committee, sorted alphabetically by organization.

> NOTE This section is generated from the machine-readable source data (see A).

Table 1 — Registered Runtime Keys

Key	Organization / Name	Description, Website, Repository
andromeda	Andromeda / Andromeda	A modern, and secure JavaScript & TypeScript runtime built in Rust, powered by the Nova engine and Oxc. Provides zero-config TypeScript, rich Web APIs (Canvas, Web Crypto, Storage, HTTP, SQLite), and aims for WinterTC compliance. Website <https://tryandromeda.dev> Repository <https://github.com/tryandromeda/andromeda>
arvancloud	Arvancloud / Edge Computing	The JavaScript runtime that powers Arvancloud Edge Computing. Website <https://www.arvancloud.ir/en/products/edge-computing>
azion	Azion / Edge Functions	Azion Edge Functions for ultra-low latency, edge-native applications, built with open standards for secure, high-performance serverless computing. Website <https://www.azion.com/en/products/edge-functions/>
bun	Oven / Bun	Incredibly fast JavaScript runtime, bundler, transpiler and package manager - all in one. Website <https://bun.sh/> Repository <https://github.com/oven-sh/bun>
convex	Convex / Convex Runtime	The JavaScript runtime used to execute deterministic Query and Mutation Functions as well as nondeterministic Action Functions in the Convex backend platform. Website <https://docs.convex.dev/functions/runtimes#default-convex-runtime> Repository <https://github.com/get-convex/convex-backend>
deno	Deno Land / Deno	A modern runtime for JavaScript and TypeScript. Website <https://deno.com> Repository <https://github.com/denoland/deno>
edge-light	Vercel / Edge Light	Developing, testing, and defining the runtime Web APIs for Edge infrastructure. Website <https://edge-runtime.vercel.app/> Repository <https://github.com/vercel/edge-runtime>
edge-routine	Alibaba Cloud / edge-routine	The JavaScript/Webassembly runtime that powers Alibaba Cloud edge-routine. Website <https://www.alibabacloud.com/help/en/dynamic-route-for-cdn/latest/er-overview>
electron	OpenJS Foundation / Electron	Build cross-platform desktop apps with JavaScript, HTML, and CSS. Website <https://www.electronjs.org/> Repository <https://github.com/electron/electron>
fastly	Fastly / JavaScript on Compute@Edge	JavaScript runtime for Fastly Compute@Edge. Website <https://developer.fastly.com/learning/compute/javascript/> Repository <https://github.com/fastly/js-compute-runtime>
kiesel	Kiesel / Kiesel	A JavaScript engine and runtime written in Zig. Website <https://kiesel.dev/> Repository <https://codeberg.org/kiesel-js/runtime>
lagon	Lagon / Lagon Runtime	Lagon is an open-source runtime and platform that allows developers to run TypeScript and JavaScript Functions at the Edge. Website <https://lagon.app> Repository <https://github.com/lagonapp/lagon>

Table 1 — Registered Runtime Keys *(continued)*

Key	Organization / Name	Description, Website, Repository
`moddable`	Moddable / Moddable SDK	Open source runtime for resource-constrained embedded devices using standard JavaScript and TypeScript. Supports standard ECMA-419 APIs. Website <https://www.moddable.com/> Repository <https://github.com/Moddable-OpenSource/moddable>
`netlify`	Netlify / Edge Functions	Edge Functions connect the Netlify platform and workflow with an open runtime standard at the network edge. Website <https://docs.netlify.com/edge-functions/overview/> Repository <https://github.com/netlify/edge-functions>
`node`	OpenJS Foundation / Node.js	Node.js is an open-source, cross-platform JavaScript runtime environment. Website <https://nodejs.org> Repository <https://github.com/nodejs/node>
`quickjs`	Fabrice Bellard / QuickJS	QuickJS is a small and embeddable Javascript engine. Website <https://bellard.org/quickjs/> Repository <https://github.com/bellard/quickjs>
`quickjs-ng`	QuickJS-NG / QuickJS-NG	A fork of the original QuickJS project by Fabrice Bellard and Charlie Gordon. Website <https://quickjs-ng.github.io/quickjs/> Repository <https://github.com/quickjs-ng/quickjs>
`pythonmonkey`	Distributive / PythonMonkey	Mozilla JS engine embedded into a Python host environment with cross-platform datatypes and event loop. Website <https://pythonmonkey.io> Repository <https://github.com/Distributive-Network/PythonMonkey>
`react-native`	Meta / React Native	A framework for building native apps using React. This key represents supported React Native JS runtimes on native platforms (excludes react-native-web). Website <https://reactnative.dev/> Repository <https://github.com/facebook/react-native>
`react-server`	React / Server Components	Used by React Server Components. Website <https://react.dev/> Repository <https://github.com/facebook/react>
`rhino`	Mozilla / Rhino	Rhino is an open-source implementation of JavaScript written entirely in Java. Website <https://rhino.github.io/> Repository <https://github.com/mozilla/rhino>
`wasmer`	Wasmer / Wasmer Edge	The JavaScript runtime that brings JavaScript to Wasmer Edge. Website <https://wasmer.io/products/edge> Repository <https://github.com/wasmerio/winterjs>
`workerd`	Cloudflare / workerd	The JavaScript / WebAssembly runtime that powers Cloudflare Workers. Website <https://workers.cloudflare.com/> Repository <https://github.com/cloudflare/workerd>

5 Governance and modification process

5.1 Proposing new runtime key entries

All ECMAScript runtimes are welcome to propose a key for inclusion in this Report, subject to the structure as described in 3. Runtimes which contribute to this Report are encouraged, but not required, to participate in Ecma TC55.

5.1.1 Proposal process

1. The proposer creates a pull request in the WinterTC55/runtime-keys GitHub repository <https://github.com/WinterTC55/runtime-keys>.
2. The pull request adds the runtime entry to the machine-readable source data file (**`runtime-keys.json`**).
3. The entry includes all mandatory metadata fields as described in 3.5.
4. The entry is inserted in alphabetical order by key.
5. The pull request is reviewed by TC55 participants.
6. At least two delegates within TC55 not directly affiliated with the proposed runtime approve the pull request.
7. The pull request is formally approved at a TC55 plenary meeting before merging.
8. After plenary approval, designated editors merge the pull request and update the current draft Report.

5.1.2 Expert review

All proposed entries undergo expert review to ensure compliance with the naming guidelines specified in 3.3 and other expectations. Reviewers evaluate whether:

- the proposed key is appropriately specific and non-generic,
- the runtime has demonstrated proof of use or adoption,
- the key does not conflict with existing entries or Browserslist identifiers, and
- the metadata is complete and accurate.

TC55 may reject a runtime key submission for any reason, including those not enumerated in the above sections. In such a scenario, the Committee will provide that reason and information on how to resolve any issue.

5.2 Modifying entries

5.2.1 Metadata modification

Modifications to runtime metadata (organization, name, description, website, or repository) follow the same process as adding entries:

1. The proposer creates a pull request modifying the entry in **`runtime-keys.json`**.
2. The proposal continues as described starting at step 5 in the approval process.

5.2.2 Key immutability

To prevent breaking existing tools and configurations that depend on runtime keys, **runtime keys are immutable** and will not be modified except in extraordinary circumstances requiring plenary consensus.

5.2.3 Deprecation

Runtime keys may be deprecated to indicate that a runtime project is inactive or discontinued. To deprecate a key:

1. The proposer creates a pull request setting the **`deprecated`** field to **`true`** in **`runtime-keys.json`**.
2. The proposal continues as described starting at step 5 in the approval process.

Deprecated keys may be undeprecated if the original runtime project resumes activity, following the same approval process. Implementations should not add support for deprecated keys.

NOTE Deprecated keys remain in the Report to preserve historical information and prevent reuse that could cause confusion.

5.3 Publication cadence

This Report is published on a semi-annual basis as an ECMA Technical Report at https://runtime-keys.proposal.wintercg.org/. Changes approved at TC55 plenary meetings between publication cycles can be found in the publicly-available draft maintained in the WinterTC runtime-keys repository.

Annex A
(normative)

Machine-readable canonical source

The authoritative, machine-readable form of the runtime key source is maintained as a JSON file: **runtime-keys.json** <https://github.com/WinterTC55/runtime-keys/blob/main/runtime-keys.json>.

A.1 JSON schema

The runtime keys source data JSON file conforms to the following structure:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Runtime Keys",
  "description": "A collection of runtime keys for ECMAScript runtime environments
  "version": "1.0.0",
  "lastModified": "YYYY-MM-DD",
  "runtimes": [
    {
      "organization": "string (required)",
      "name": "string (required)",
      "key": "string (required)",
      "description": "string (required)",
      "website": "string | null (optional)",
      "repository": "string | null (optional)",
      "deprecated": "boolean (required, default: false)"
    }
  ]
}
```

A.2 Accessing the source data

The current machine-readable source is available at:

- Draft version:
  https://github.com/WinterTC55/runtime-keys/blob/main/runtime-keys.json
- Published version: To be determined upon Ecma TR publication

Annex B
(informative)

Example usage

This Annex provides informative examples of how runtime keys may be used in practice.

> NOTE This specification does not prescribe how runtime keys should be used. The following examples are illustrative only.

B.1 package.json example

A library package might use runtime keys in its **`package.json`** to specify conditional exports and runtime version requirements:

```
{
  "name": "example-library",
  "version": "1.0.0",
  "exports": {
    "node": "./dist/node/index.js",
    "deno": "./dist/deno/index.js",
    "bun": "./dist/bun/index.js",
    "workerd": "./dist/workerd/index.js"
  },
  "engines": {
    "node": ">=18",
    "deno": ">=1.30",
    "bun": ">=1.0"
  }
}
```

B.2 Configuration File Example

A build tool or framework might use runtime keys to specify runtime-specific configuration:

```
{
  "runtimes": {
    "node": {
      "outputDir": "./dist/node",
      "target": "ES2022"
    },
    "deno": {
      "outputDir": "./dist/deno",
      "target": "ES2022"
    },
    "workerd": {
      "outputDir": "./dist/workerd",
      "target": "ES2022",
      "polyfills": ["node:buffer"]
    }
  }
}
```

Bibliography

ECMA-262, *ECMAScript® Language Specification*
https://tc39.es/ecma262/multipage/

Software License

Ecma International
Rue du Rhone 114
CH-1204 Geneva
Tel: +41 22 849 6000
Fax: +41 22 849 6001
Web: https://ecma-international.org/