SerializedGraphJson

August 18, 2026

SerializedGraphJson

Kind: Interface

Source: `packages/core/inspector/interfaces/serialized-graph-json.interface.ts`

Part of: Core

`SerializedGraphJson` defines the JSON-ready representation of an inspected graph. It stores graph nodes, edges, entrypoints, supplemental data, execution status, and metadata in a normalized structure suitable for serialization, transport, and visualization.

Properties

Property	Type
`nodes`	`Record<string, Node>`
`edges`	`Record<string, Edge>`
`entrypoints`	`Record<string, Entrypoint<unknown>[]>`
`extras`	`Extras`
`status`	`SerializedGraphStatus`
`metadata`	`SerializedGraphMetadata`

Diagram

```
graph LR
  Graph[SerializedGraphJson]
  Graph --> Nodes["nodes: Record<string, Node>"]
  Graph --> Edges["edges: Record<string, Edge>"]
  Graph --> Entrypoints["entrypoints: Record<string, Entrypoint<unknown>[]>"]
  Graph --> Extras["extras: Extras"]
  Graph --> Status["status: SerializedGraphStatus"]
  Graph --> Metadata["metadata: SerializedGraphMetadata"]
```

```
  Edges --> Nodes
  Entrypoints --> Nodes
```

Usage

```ts
import type { SerializedGraphJson } from './interfaces/serialized-graph-json.interface';

const serializedGraph: SerializedGraphJson = {
  nodes: {
    start: {
      id: 'start',
      type: 'input',
    },
    process: {
      id: 'process',
      type: 'transform',
    },
  },
  edges: {
    'start-to-process': {
      id: 'start-to-process',
      source: 'start',
      target: 'process',
    },
  },
  entrypoints: {
    default: [
      {
        nodeId: 'start',
      },
    ],
  },
  extras: {},
  status: 'ready',
  metadata: {
    name: 'Example graph',
  },
};

// Serialize for storage, inspector transport, or UI rendering.
const json = JSON.stringify(serializedGraph);
```

AI Coding Instructions

- Keep `nodes` and `edges` keyed by stable, unique IDs; ensure edge source and target references match node IDs.

- Preserve the normalized record-based structure rather than converting graph data into arrays unless a consuming API explicitly requires it.
- Treat `entrypoints` as grouped entrypoint definitions; support multiple `Entrypoint<unknown>` values per entrypoint key.
- Populate `status` and `metadata` consistently so inspector consumers can determine graph state and display graph context.
- Use JSON-serializable values in `extras` and metadata fields when this object will be persisted or sent across process boundaries.

How it works

`SerializedGraphJson` is the exported TypeScript interface for the object returned by `SerializedGraph.toJSON()`. It describes a serialized graph as four required collections plus optional status and diagnostic metadata. [serialized-graph-json.interface.ts:8-15] [serialized-graph.ts:125-140]

- `nodes` is a string-keyed record of `Node` values. A node has `id` and `label`, and represents either a module or a class-related item with its corresponding metadata. [serialized-graph-json.interface.ts:9] [node.interface.ts:4-47]
- `edges` is a string-keyed record of `Edge` values. Each edge has an `id`, `source`, `target`, and metadata describing either a module-to-module or class-to-class connection. [serialized-graph-json.interface.ts:10] [edge.interface.ts:8-31]
- `entrypoints` maps a string parent ID to an array of entrypoints. Each entrypoint records its type, method and class names, class-node ID, and metadata containing a `key`; `id` is optional. [serialized-graph-json.interface.ts:11] [entrypoint.interface.ts:17-24]
- `extras` contains arrays for orphaned enhancers (`subtype` and `ref`) and attached enhancers (`nodeId`). [serialized-graph-json.interface.ts:12] [extras.interface.ts:6-21]
- `status`, when present, is either `'partial'` or `'complete'`. [serialized-graph-json.interface.ts:13] [serialized-graph.ts:22]
- `metadata`, when present, contains a `cause` whose type is `'unknown-dependencies'` or `'unknown'`, with optional dependency context, module ID, node ID, and error fields. [serialized-graph-json.interface.ts:14] [serialized-graph-metadata.interface.ts:3-11]

`SerializedGraph.toJSON()` converts its internal node, edge, and entrypoint `Map` instances into the corresponding records with `Object.fromEntries`, and assigns its `extras` object directly to the result. [serialized-graph.ts:26-35] [serialized-

graph.ts:125-131] It adds `status` when `_status` is truthy and `metadata` when `_metadata` is truthy; `_status` starts as `'complete'`, while metadata starts unset. [serialized-graph.ts:34-35] [serialized-graph.ts:133-139]

The interface itself contains no runtime validation, error handling, or side effects; it only declares the object shape. [serialized-graph-json.interface.ts:8-15]