SerializedGraph

August 18, 2026

SerializedGraph

Kind: Class

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

Part of: Core

`SerializedGraph` builds an inspector-friendly representation of a graph by collecting nodes, edges, entrypoints, and enhancer relationships. It provides lookup support during graph construction and exposes `toJSON()` and `toString()` for exporting the completed graph to tooling, logs, or persisted inspector output.

Methods

Method	Signature	Returns
`insertNode`	`insertNode(nodeDefinition: Node)`	`void`
`insertEdge`	`insertEdge(edgeDefinition: WithOptionalId<Edge>)`	`void`
`insertEntrypoint`	`insertEntrypoint(definition: Entrypoint<T>, parentId: string)`	`void`
`insertOrphanedEnhancer`	`insertOrphanedEnhancer(entry: OrphanedEnhancerDefinition)`	`void`
`insertAttachedEnhancer`	`insertAttachedEnhancer(nodeId: string)`	`void`
`getNodeById`	`getNodeById(id: string)`	`void`
`toJSON`	`toJSON()`	`SerializedGraphJson`
`toString`	`toString()`	`void`

Where it refuses work

- `SerializedGraph` stops the work with an early return when `this.nodes.has(nodeDefinition.id)`.
- `SerializedGraph` stops the work with an early return when `typeof value === 'symbol'`.

Diagram

```
graph LR
  Entrypoint[Entrypoint] --> NodeA[Serialized Node]
  NodeA -->|edge| NodeB[Serialized Node]
  NodeA -->|attached enhancer| EnhancerA[Enhancer]
  OrphanedEnhancer[Orphaned Enhancer] --> Graph[SerializedGraph]
  NodeB --> Graph
  EnhancerA --> Graph
  Graph --> JSON[toJSON()]
  Graph --> Text[toString()]
```

Usage

```
import { SerializedGraph } from './serialized-graph';

const graph = new SerializedGraph();

// Add graph entities using the inspector's serialized descriptor types.
graph.insertNode({
  id: 'user-service',
  label: 'UserService',
});

graph.insertNode({
  id: 'user-repository',
  label: 'UserRepository',
});

graph.insertEdge({
  from: 'user-service',
  to: 'user-repository',
});

graph.insertEntrypoint({
  id: 'get-user',
  nodeId: 'user-service',
});

graph.insertAttachedEnhancer({
  nodeId: 'user-service',
  enhancer: {
    id: 'auth-guard',
```

```
    type: 'guard',
  },
});

// Export the completed graph for inspector consumers.
const serialized = graph.toJSON();
console.log(serialized);

// Use the string form for diagnostics or snapshots.
console.log(graph.toString());

// Look up a previously inserted node while building relationships.
const serviceNode = graph.getNodeById('user-service');
```

AI Coding Instructions

- Insert nodes before adding edges or attached enhancers that reference their IDs; use stable, unique IDs throughout graph construction.
- Use `getNodeById()` when extending an existing node rather than duplicating node records.
- Represent enhancers attached to a known node with `insertAttachedEnhancer()`; use `insertOrphanedEnhancer()` only when no owning node can be resolved.
- Treat `toJSON()` as the structured integration boundary for inspector consumers, snapshots, and transport layers.
- Keep `toString()` usage limited to debugging and diagnostics; prefer `toJSON()` for programmatic processing.

How it works

`SerializedGraph` is a mutable in-memory representation of an inspected application graph. It stores nodes and edges by ID, entrypoints grouped by parent ID, enhancer-related extras, a graph status, and optional failure metadata. Its initial status is `'complete'`; its extras start with empty orphaned- and attached-enhancer arrays. `packages/core/inspector/serialized-graph.ts:26-35`

- A node has an `id` and `label`, and is either:
  - a module node with `global`, `dynamic`, and `internal` metadata; or
  - a class node—provider, controller, middleware, or injectable—with its parent module ID and class-related metadata such as token, scope, lifecycle flags, export state, and initialization time.
    `packages/core/inspector/interfaces/node.interface.ts:4-47`

- An edge connects `source` and `target` node IDs. Edge metadata describes either a module import connection or a class-to-class dependency, including injection type and, where applicable, the constructor parameter, property, or decorator key/index. `packages/core/inspector/interfaces/edge.interface.ts:3-31`
- Entrypoints are stored as arrays under a caller-supplied parent ID. Each entrypoint includes its type, method name, class name, class node ID, and metadata with a `key`. `packages/core/inspector/serialized-graph.ts:29` `packages/core/inspector/serialized-graph.ts:102-109` `packages/core/inspector/interfaces/entrypoint.interface.ts:17-24`

`insertNode(node)` marks provider nodes as `internal: true` when their token is one of the class's listed framework-internal tokens, such as `ApplicationConfig`, `ModuleRef`, `HttpAdapterHost`, `LazyModuleLoader`, `ExternalContextCreator`, `ModulesContainer`, `Reflector`, `SerializedGraph`, `REQUEST`, or `INQUIRER`. `packages/core/inspector/serialized-graph.ts:37-50` `packages/core/inspector/serialized-graph.ts:60-69` It then inserts the node only when its ID is absent; for an existing ID, it returns the already stored node rather than replacing it. The internal-marker mutation occurs before that duplicate check. `packages/core/inspector/serialized-graph.ts:60-75`

`insertEdge(edge)` accepts an edge with an optional ID. For a class-to-class edge, it marks the edge metadata as internal when either class token is in that same internal-token list. `packages/core/inspector/serialized-graph.ts:77-91` When no ID is supplied, it serializes the edge definition with `JSON.stringify` and passes the resulting string to `DeterministicUuidRegistry.get()`. `packages/core/inspector/serialized-graph.ts:92-99` The registry derives an ID from a 31-based integer hash and retries with an incremented suffix if that ID is already registered. `packages/core/inspector/deterministic-uuid-registry.ts:4-10` The completed edge is stored under its ID, replacing any edge currently stored under that ID, and is returned. `packages/core/inspector/serialized-graph.ts:94-99`

`insertEntrypoint(definition, parentId)` appends the definition to the existing collection for `parentId`, or creates a one-item collection when none exists. `packages/core/inspector/serialized-graph.ts:102-109` `insertOrphanedEnhancer(entry)` appends an enhancer definition to `extras.orphanedEnhancers`, while `insertAttachedEnhancer(nodeId)` appends `{ nodeId }` to `extras.attachedEnhancers`; neither method removes duplicates. `packages/core/inspector/serialized-graph.ts:111-119` The corresponding types identify orphaned enhancers by subtype and reference, and attached enhancers by node ID. `packages/core/inspector/interfaces/extras.interface.ts:3-21`

`getNodeById(id)` returns the node associated with `id`, or the map's missing-value result when no such node exists. `packages/core/inspector/serialized-graph.ts:121-123`

The `status` and `metadata` setters directly replace their respective private fields. `packages/core/inspector/serialized-graph.ts:52-58` Status is limited by its TypeScript type to `'partial'` or `'complete'`. `packages/core/inspector/serialized-graph.ts:22` Metadata records either an `unknown-dependencies` cause with optional dependency context, module ID, and node ID, or an `unknown` cause with an optional error value. `packages/core/inspector/interfaces/serialized-graph-metadata.interface.ts:3-11`

`toJSON()` converts the node, edge, and entrypoint maps into plain object records, retains the `extras` object by reference, always includes the initialized status, and adds metadata only when it is set. `packages/core/inspector/serialized-graph.ts:125-140` `toString()` JSON-formats that result with two-space indentation; its replacer converts symbol values to their string form and function values to their name, falling back to `'Function'`. `packages/core/inspector/serialized-graph.ts:142-150`

The container owns one graph instance and exposes it through `serializedGraph`. `packages/core/injector/container.ts:31-40` The internal core module also registers that same instance under the `SerializedGraph` injection token. `packages/core/injector/internal-core-module/internal-core-module-factory.ts:69-72` `GraphInspector` populates it from modules, class wrappers, imports, dependency metadata, enhancers, and entrypoint definitions. `packages/core/inspector/graph-inspector.ts:22-36` [`packages/core/inspector/graph-inspector.ts:100-150](packages/core/inspector/graph-inspector.ts#L100-L150) On inspection failure, GraphInspector.registerPartial() sets the graph status to 'partial', records cause metadata, and registers this graph with PartialGraphHost . [packages/core/inspector/graph-inspector.ts:38-59`](packages/core/inspector/graph-inspector.ts#L38-L59)

There is no explicit runtime input validation or class-defined error handling in these mutation and serialization methods; callers must supply values whose accessed fields match the declared node, edge, entrypoint, enhancer, status, and metadata shapes. `packages/core/inspector/serialized-graph.ts:52-155`