

If you want to **fully qualify a type name** in TypeScript — for example, to distinguish between `Map` from the global ES standard library and a `Map` type you've defined in your own module — you can do it in two main ways, depending on whether you're talking about **runtime imports** or **type-level qualification**.

## 1. Fully Qualifying a Type via Namespace or Module Import

If the type comes from a module, you can import it under a namespace and reference it with a fully qualified name:

```
import * as std from "typescript"; // Example for TS Compiler API types

// Fully qualified type usage
let node: std.Node;
```

For built-in types like `Map<K, V>` from the standard library, you can explicitly reference them via the `global` namespace:

```
// Explicitly using the global Map type
let myMap: globalThis.Map<string, number> = new Map();
```

Here:

- `globalThis.Map` ensures you're referring to the **built-in** `Map` type, not a locally declared one.

## 2. Using the TypeScript Compiler API to Get Fully Qualified Names

If you're writing tooling (e.g., a linter, code generator, or analyzer), you can use the **TypeScript Compiler API** to retrieve a type's fully qualified name.

Example:

```
import ts from "typescript";

function getFullyQualifiedTypeName(fileName: string, typeName: string) {
```

```
const program = ts.createProgram([fileName], {});
const checker = program.getTypeChecker();

const sourceFile = program.getSourceFile(fileName);
if (!sourceFile) throw new Error("Source file not found");

ts.forEachChild(sourceFile, function visit(node) {
    if (ts.isTypeAliasDeclaration(node) ||
ts.isInterfaceDeclaration(node) || ts.isClassDeclaration(node)) {
        if (node.name?.text === typeName) {
            const symbol =
checker.getSymbolAtLocation(node.name);
            if (symbol) {
                const fqn =
checker.getFullyQualifiedName(symbol);
                console.log(`Fully qualified name: ${fqn}`);
            }
        }
        ts.forEachChild(node, visit);
    });
}

// Usage example
getFullyQualifiedTypeName("example.ts", "Map");
```

This will print something like:

"global.Map"

or, for a module type:

"\`@angular/platform-browser\``.DomSanitizer"

### Key Points:

- Use `globalThis.Map` to explicitly reference the built-in `Map` type.
- Use `import * as ns` and `ns.TypeName` for module-scoped types.

- Use `TypeChecker.getFullyQualifiedName(symbol)` in the Compiler API for tooling.

If you want, I can give you a **ready-to-run TypeScript script** that prints the fully qualified name of **any type** in a file using the Compiler API.

Do you want me to prepare that?