schani Merge pull request #1697 from timia2109/csharp-virtual-proper… … daffe53 on 15 Jul 2021 | 3,448 commits

.buildkite	Add javascript-prop-types to pipeline	2 years ago
.vscode	Fix an import	4 years ago
appcenter	Move app.quicktype.io CI to Buildkite	4 years ago
build	build-utils: add missing LICENSE	10 months ago
data	TypeScript input from strings	4 years ago
doc	Documentation	4 years ago
media/faq	Add screenshots for the FAQ	4 years ago
script	environmental variable fix for #1459	2 years ago
src	Merge pull request #1697 from timia2109/csharp-virtual-properties	6 months ago
test	Skip postman-collection.schema test as well	2 years ago
.deepsource.toml	Deepsource config	10 months ago
.dockerignore	CI improvements	4 years ago
.gitattributes	Create .gitattributes	2 years ago
.gitignore	Java OffsetDateTime working	2 years ago
Dockerfile	Merge pull request #1669 from quicktype/use-old-dart	11 months ago
FAQ.md	Explain how quicktype differs from other JSON converters	2 years ago
LICENSE	Create LICENSE	5 years ago
PACKAGES.md	Auto increment package versions	4 years ago
README.md	Fix programmatic example in README ("Calling `quicktype` from Ja…	13 months ago
package-lock.json	Merge pull request #1716 from quicktype/dependabot/npm_and_yar…	6 months ago
package.json	Bump lodash from 4.17.19 to 4.17.21	9 months ago
quicktype-logo.svg	More appropriate README content for GitHub (#352)	4 years ago
tslint.json	Be more strict about async and promises	4 years ago

README.md

quicktype

npm package 15.0.260 | build passing

`quicktype` generates strongly-typed models and serializers from JSON, JSON Schema, TypeScript, and GraphQL queries, making it a breeze to work with JSON type-safely in many programming languages.

- Try `quicktype` in your browser.
- Read 'A first look at quicktype' for more introduction.
- If you have any questions, check out the FAQ first.

Supported Inputs

JSON	JSON API URLs	JSON Schema

TypeScript	GraphQL queries

Target Languages

Ruby	JavaScript	Flow	Rust	Kotlin

Dart	Python	C#	Go	C++

Java	TypeScript	Swift	Objective-C	Elm

JSON Schema	Pike	Prop-Types	Haskell	

Missing your favorite language? Please implement it!

Installation

There are many ways to use `quicktype`. app.quicktype.io is the most powerful and complete UI. The web app also works offline and doesn't send your sample data over the Internet, so paste away!

For the best CLI, we recommend installing `quicktype` globally via `npm`:

```
npm install -g quicktype
```

Other options:

- Homebrew *(infrequently updated)*
- Xcode extension*
- VSCode extension*
- Visual Studio extension*

* limited functionality

Using `quicktype`

```
# Run quicktype without arguments for help and options
quicktype

# quicktype a simple JSON object in C#
echo '{ "name": "David" }' | quicktype -l csharp

# quicktype a top-level array and save as Go source
echo '[1, 2, 3]' | quicktype -o ints.go

# quicktype a sample JSON file in Swift
quicktype person.json -o Person.swift

# A verbose way to do the same thing
quicktype \
  --src person.json \
  --src-lang json \
  --lang swift \
  --top-level Person \
  --out Person.swift

# quicktype a directory of samples as a C++ program
# Suppose ./blockchain is a directory with files:
#   latest-block.json transactions.json marketcap.json
quicktype ./blockchain -o blockchain-api.cpp

# quicktype a live JSON API as a Java program
quicktype https://api.somewhere.com/data -o Data.java
```

Generating code from JSON schema

The recommended way to use `quicktype` is to generate a JSON schema from sample data, review and edit the schema, commit the schema to your project repo, then generate code from the schema as part of your build process:

```
# First, infer a JSON schema from a sample.
quicktype pokedex.json -l schema -o schema.json

# Review the schema, make changes,
# and commit it to your project repo.

# Finally, generate model code from schema in your
# build process for whatever languages you need:
quicktype -s schema schema.json -o src/ios/models.swift
quicktype -s schema schema.json -o src/android/Models.java
quicktype -s schema schema.json -o src/nodejs/Models.ts

# All of these models will serialize to and from the same
# JSON, so different programs in your stack can communicate
# seamlessly.
```

Generating code from TypeScript (Experimental)

You can achieve a similar result by writing or generating a TypeScript file, then quicktyping it. TypeScript is a typed superset of JavaScript with simple, succinct syntax for defining types:

```
interface Person {
  name: string;
  nickname?: string; // an optional property
  luckyNumber: number;
}
```

You can use TypeScript just like JSON schema was used in the last example:

```
# First, infer a TypeScript file from a sample (or just write one!)
quicktype pokedex.json -o pokedex.ts --just-types
# Review the TypeScript, make changes, etc.
quicktype pokedex.ts -o src/ios/models.swift
```

Calling `quicktype` from JavaScript

You can use `quicktype` as a JavaScript function within `node` or browsers. First add the `quicktype-core` package:

```
$ npm install quicktype-core
```

In general, first you create an `InputData` value with one or more JSON samples, JSON schemas, TypeScript sources, or other supported input types. Then you call `quicktype`, passing that `InputData` value and any options you want.

```
const {
  quicktype,
  InputData,
  jsonInputForTargetLanguage,
  JSONSchemaInput,
  FetchingJSONSchemaStore,
} = require("quicktype-core");

async function quicktypeJSON(targetLanguage, typeName, jsonString) {
  const jsonInput = jsonInputForTargetLanguage(targetLanguage);

  // We could add multiple samples for the same desired
  // type, or many sources for other types. Here we're
  // just making one type from one piece of sample JSON.
  await jsonInput.addSource({
    name: typeName,
    samples: [jsonString],
  });

  const inputData = new InputData();
  inputData.addInput(jsonInput);

  return await quicktype({
    inputData,
    lang: targetLanguage,
  });
}

async function quicktypeJSONSchema(targetLanguage, typeName, jsonSchemaString) {
  const schemaInput = new JSONSchemaInput(new FetchingJSONSchemaStore());

  // We could add multiple schemas for multiple types,
  // but here we're just making one type from JSON schema.
  await schemaInput.addSource({ name: typeName, schema: jsonSchemaString });

  const inputData = new InputData();
  inputData.addInput(schemaInput);

  return await quicktype({
    inputData,
    lang: targetLanguage,
  });
}

async function main() {
  const { lines: swiftPerson } = await quicktypeJSON(
    "swift",
    "Person",
    jsonString
  );
  console.log(swiftPerson.join("\n"));

  const { lines: pythonPerson } = await quicktypeJSONSchema(
    "python",
    "Person",
    jsonSchemaString
  );
  console.log(pythonPerson.join("\n"));
}

main();
```

The argument to `quicktype` is a complex object with many optional properties. Explore its definition to understand what options are allowed.

Contributing

`quicktype` is Open Source and we love contributors! In fact, we have a list of issues that are low-priority for us, but for which we'd happily accept contributions. Support for new target languages is also strongly desired. If you'd like to contribute, need help with anything at all, or would just like to talk things over, come join us on Slack.

Setup, Build, Run

`quicktype` is implemented in TypeScript and requires `nodejs` and `npm` to build and run.

First, install `typescript` globally via `npm`:

Clone this repo and do:

macOS / Linux

```
npm install
script/quicktype # rebuild (slow) and run (fast)
```

Windows

```
npm install --ignore-scripts # Install dependencies
npm install -g typescript # Install typescript globally
tsc --project src/cli # Rebuild
node dist\cli\index.js # Run
```

Edit

Install Visual Studio Code, open this workspace, and install the recommended extensions:

```
code . # opens in VS Code
```

Live-reloading for quick feedback

When working on an output language, you'll want to view generated output as you edit. Use `npm start` to watch for changes and recompile and rerun `quicktype` for live feedback. For example, if you're developing a new renderer for `fortran`, you could use the following command to rebuild and reinvoke `quicktype` as you implement your renderer:

```
npm start -- "--lang fortran pokedex.json"
```

The command in quotes is passed to `quicktype`, so you can render local `.json` files, URLs, or add other options.

Test

`quicktype` has many complex test dependencies:

- `crystal` compiler
- `dotnetcore` SDK
- Java, Maven
- `elm` tools
- `g++` C++ compiler
- `golang` stack
- `swift` compiler
- `clang` and Objective-C Foundation (must be tested separately on macOS)
- `rust` tools
- `pike` interpreter
- Bundler for Ruby
- `haskell` stack

We've assembled all of these tools in a Docker container that you build and test within:

```
# Build and attach to Docker container
script/dev

# Run full test suite
npm run test

# Test a specific language (see test/languages.ts)
FIXTURE=golang npm test

# Test a single sample or directory
FIXTURE=swift npm test -- pokedex.json
FIXTURE=swift npm test -- test/inputs/json/samples
```

About

Generate types and converters from JSON, Schema, and GraphQL

app.quicktype.io

kotlin java graphql swift rust golang json typescript csharp cplusplus objective-c json-schema elm

Readme
Apache-2.0 License
7.6k stars
84 watching
677 forks

Releases

6 tags

Packages

No packages published

Used by 347

+ 339

Contributors 83

+ 72 contributors

Environments 1

github-pages Active

Languages

- TypeScript 97.7%
- Python 0.7%
- Shell 0.7%
- JavaScript 0.7%
- Dockerfile 0.2%
- PowerShell 0.0%