FlatBuffers Documentation

Overview

FlatBuffers is an efficient cross platform serialization library for C++, C#, C, Go, Java, JavaScript, PHP, and Python. It was originally created at Google for game development and other performance-critical applications.

It is available as Open Source on GitHub under the Apache license, v2 (see LICENSE.txt).

Why use FlatBuffers?

- **Access to serialized data without parsing/unpacking** - What sets FlatBuffers apart is that it represents hierarchical data in a flat binary buffer in such a way that it can still be accessed directly without parsing/unpacking, while also still supporting data structure evolution (forwards/backwards compatibility).
- **Memory efficiency and speed** - The only memory needed to access your data is that of the buffer. It requires 0 additional allocations (in C++, other languages may vary). FlatBuffers is also very suitable for use with mmap (or streaming), requiring only part of the buffer to be in memory. Access is close to the speed of raw struct access with only one extra indirection (a kind of vtable) to allow for format evolution and optional fields. It is aimed at projects where spending time and space (many memory allocations) to be able to access or construct serialized data is undesirable, such as in games or any other performance sensitive applications. See the benchmarks for details.
- **Flexible** - Optional fields means not only do you get great forwards and backwards compatibility (increasingly important for long-lived games: don't have to update all data with each new version!). It also means you have a lot of choice in what data you write and what data you don't, and how you design data structures.
- **Tiny code footprint** - Small amounts of generated code, and just a single small header as the minimum dependency, which is very easy to integrate. Again, see the benchmark section for details.
- **Strongly typed** - Errors happen at compile time rather than manually having to write repetitive and error prone run-time checks. Useful code can be generated for you.
- **Convenient to use** - Generated C++ code allows for terse access & construction code. Then there's optional functionality for parsing schemas and JSON-like text representations at runtime efficiently if needed (faster and more memory efficient than other JSON parsers).

  Java and Go code supports object-reuse. C# has efficient struct based accessors.
- **Cross platform code with no dependencies** - C++ code will work with any recent gcc/clang and VS2010. Comes with build files for the tests & samples (Android .mk files, and cmake for all other platforms).

Why not use Protocol Buffers, or .. ?

Protocol Buffers is indeed relatively similar to FlatBuffers, with the primary difference being that FlatBuffers does not need a parsing/ unpacking step to a secondary representation before you can access data, often coupled with per-object memory allocation. The code is an order of magnitude bigger, too. Protocol Buffers has neither optional text import/export nor schema language features like unions.

But all the cool kids use JSON!

JSON is very readable (which is why we use it as our optional text format) and very convenient when used together with dynamically typed languages (such as JavaScript). When serializing data from statically typed languages, however, JSON not only has the obvious drawback of runtime inefficiency, but also forces you to write *more* code to access data (counterintuitively) due to its dynamic-typing serialization system. In this context, it is only a better choice for systems that have very little to no information ahead of time about what data needs to be stored.

If you do need to store data that doesn't fit a schema, FlatBuffers also offers a schema-less (self-describing) version!

Read more about the "why" of FlatBuffers in the white paper.

Who uses FlatBuffers?

- Cocos2d-x, the #1 open source mobile game engine, uses it to serialize all their game data.
- Facebook uses it for client-server communication in their Android app. They have a nice article explaining how it speeds up loading their posts.
- Fun Propulsion Labs at Google uses it extensively in all their libraries and games.

Usage in brief

This section is a quick rundown of how to use this system. Subsequent sections provide a more in-depth usage guide.

- Write a schema file that allows you to define the data structures you may want to serialize. Fields can have a scalar type (ints/floats of all sizes), or they can be a: string; array of any type; reference to yet another object; or, a set of possible objects (unions). Fields are optional and have defaults, so they don't need to be present for every object instance.
- Use `flatc` (the FlatBuffer compiler) to generate a C++ header (or Java/C#/Go/Python.. classes) with helper classes to access and construct serialized data. This header (say `mydata_generated.h`) only depends on `flatbuffers.h`, which defines the core functionality.
- Use the `FlatBufferBuilder` class to construct a flat binary buffer. The generated functions allow you to add objects to this buffer recursively, often as simply as making a single function call.
- Store or send your buffer somewhere!
- When reading it back, you can obtain the pointer to the root object from the binary buffer, and from there traverse it conveniently in-place with `object->field()`.

In-depth documentation

- How to build the compiler and samples on various platforms.
- How to use the compiler.
- How to write a schema.
- How to use the generated C++ code in your own programs.
- How to use the generated Java/C# code in your own programs.
- How to use the generated Go code in your own programs.
- How to use FlatBuffers in C with `flatcc` in your own programs.
- Support matrix for platforms/languages/features.
- Some benchmarks showing the advantage of using FlatBuffers.
- A white paper explaining the "why" of FlatBuffers.
- How to use the schema-less version of FlatBuffers.
- A description of the internals of FlatBuffers.
- A formal grammar of the schema language.

Online resources

- GitHub repository
- Landing page
- FlatBuffers Google Group
- FlatBuffers Issues Tracker
- Independent implementations & tools:
    - FlatCC Alternative FlatBuffers parser, code generator and runtime all in C.
- Videos:
    - Colt's DevByte.
    - GDC 2015 Lightning Talk.
    - FlatBuffers for Go.
    - Evolution of FlatBuffers visualization.
- Useful documentation created by others:
    - FlatBuffers in Go
    - FlatBuffers in Android
    - Parsing JSON to FlatBuffers in Java
    - FlatBuffers in Unity