Project Common Flags

Introduction

Name of Document Author
Brett Lawson <brett19@gmail.com>

Date of this Document
Friday, Sept 5, 2014

Proposal Status
ACCEPTED

Name of Major Document Consumers
SDK Client Developers

Project Summary
This project aims to provide a consistent method for storing document meta-data. Specifically it will define the exact formatting of the flags field as well as the datatype field to allow the server to make reasonable assumptions as well. This document additionally attempts to define a set of 'universally supported' document formats. Bits are currently set aside for compression in this proposal, but it does not constrain how those bits may be used and a future extension to this proposal may choose to use them differently.

Risks and Assumptions
This proposal assumes that no clients currently abuse the top 8 bits of the flags object, this has been confirmed to be the case with all existing memcached and couchbase clients, but may not be the case for any clients developed by third-parties. It should be noted this will not break anything for those third-party clients; it merely won't solve any interoperability issues for them. Additionally, it is assumed that any consumer of the flags data (customers with custom transcoders) will continue to use custom transcoders with the new SDKs which implement this proposal, and thus will be a non-concern as the proposals logic will not be concerned with these implementations.

Business Summary

Problem Area
Currently, there is no defined method by which multiple SDKs or the clients can coordinate. This means there is a fair chance that documents stored by one SDK may not be able to be retrieved by another.

Market / Requester

Customers who are using more than one SDK, such is often the case with tools being written vs the application itself.

How will you know you are done?

All clients will speak one universal language in regards to meta data. Documents stored with one SDK will be retrievable and manipulable by any other SDK.

Technical Description

Details

This proposal will specify a format for the upper 8 bits of the flags field for common use among clients and allow for client-specific data in the lower 16 bits (the middle 8 bits are reserved for future use, or possibly backwards compatibility use if necessary).

During reading, the client should check these upper 8 bits for a non-zero value, and if this is the case, the common flags format should be used. If these upper 8 bits are zeroed, the clients should fall back to the existing logic used in the respective client.
When writing, the client should set the upper 8 bits according to the format below, and additionally set the lowest 16 bits according to the existing logic that was used in their client.

If a client encounters a format or compression type which is unknown, an error should be propagated to the user advising them that the data is stored with an unknown format. Additionally, any bits which are marked as reserved must be zero, an error should be propagated if this is not the case.

The format of the upper 8 bits is as follows:

> The top 3 bits are used to specify the type of compression that is being used, the lower 4 bits are used to define the format, the middle 1 bit is currently reserved to allow expanding of compression or format fields as needed. The following is a list of all supposed formats, note that the ‘private’ format is used for sdk-specific encodings and the ‘reserved’ format is used to avoid a completely zeroed upper 8 bits which would interfere with detecting the presence of common flags.
>
> flags (32 bits):
> `<compression:3> 0 <format:4> | ? ? ? ? ? ? ? ? | ? ? ? ? ? ? ? ? | ? ? ? ? ? ? ? ?`
>
> *Formats (All must be supported):*
> 0 - Reserved
> 1 - Private
> 2 - JSON
> 3 - Raw Binary

4 - Non-JSON String (utf-8, no bom)

Compressions (expected compatibility specified individually below):
0 - None (MUST be supported)

Special Note: Pure numeric values could be encoded as both utf-8 strings as well as JSON. These values should always be encoded as JSON for the purpose of the common flags format.

Related Tickets
None.

Other Solutions Previously Considered
The possible use of the datatype field was originally considered, but due to an unrelated implementation, that proposal is no longer valid.

In Scope
All Client Libraries (except libcouchbase, which does not perform value encoding, though all tools should still understand this specification)

Out of Scope
All Server Components

Interfaces
No public facing interfaces should be affect by this change. However, clients SHOULD provide a custom transcoder interface to ensure any edge-cases have a workaround, this may affect public facing interfaces.

Doc Impact
The change to the internal representation of flags field MUST be documented clearly for customers. This is especially important as data that was stored in the old format will end up being ‘on the fly’ converted and may not be fully compatible with old memcached clients without implementing a custom transcoder.

Packaging and Delivery Impact
No impact. This proposal is attached to the SDK 2.0 proposal, which already encompasses all necessary changes.

Security Impact
No impact.

Dependencies
There is a soft dependency on the SDK 2.0 changes as this project is intended to be a sub-proposal of the SDK 2.0 proposal (mainly to wrap all behaviour changes together into a

single release). However, this proposals changes themselves do not depend on any SDK 2.0 proposal changes.

Reference Documents

Please additionally refer to the Server DataType changes document for information on how to accurately store meta data in the datatype field for server consumption.

Open Questions

Resolved Questions

- Q: Heuristic compression detection, or part of datatype bitfield?
  A: Heuristic compression detection will not be used.

- Q: Spymemcached backwards-compatibility?
  A: Spymemcached correctly reads the flags data as a bitfield, preventing any compatibility issues.

- Q: Should we support primitive datatypes?
  A: 64-bit signed integers, doubles will be supported. (spec updated)

- Q: Which formats/compressions must be supported by each SDK?
  A: All formats; compressions are SDK specific at the moment. (spec updated)

- Q: Should tools be updated to display the type information stored in flags?
  A: Yes, but the timeline on this is currently uncertain. (spec updated)

- Q: Should snappy be part of common flags since it is now a direct server feature?
  A: Probably not, however it is a support compression from a client-library perspective, and having this as a known value allows the user to pass already-compressed data to our libraries and avoid double-compression and compression overhead.

- Q: Endianness and exact representation should be defined for primitive data types.
  A: The spec was updated to define these.