Universal File Format and Supporting Infrastructure

Marshall, Josh
joshua.r.marshall.1991@gmail.com

March 29, 2020

Abstract

Similar to how the most complete meaning of a 'function' is deceptively complex, so too is the handling of computer files. This fundamental process of serializing and deserializing between active memory, and a format suitable for non-volatile storage and transmission has incurred massive costs over time. Many programs have developed custom serializers and deserializers. This has resulted in disunity of file formats which has in turn led to encoding inefficiencies, lost backwards and forwards compatibility, portability failures, and redundant work. Performing this work for each program is expensive. This paper proposes and formalizes a new model to file handling to ease developer burden, increase space efficiency, improve system integration, allow for much greater compatibility, robust security, transparent sharding across disparate systems, and future extensibility while lowering overall developer burden.

1 Introduction

This paper is written in LaTeX, compiled to a pdf, and will likely be read online in HTML, Markdown, or ReStructured Text. They will all encode the same fundamental information in mutually incompatible ways. Yet, a system accessing these formats will still display them similarly. In system memory, their decoded state can be made, in theory, identical. This is not to say that any of these formats should replace all of the others, but it does exemplify the complexity and divergence in performing a similar simple task in computing.

Imagine the workflow for writing this document – a program runs where a user enters some data into memory through a text editor, that memory is transformed by some serialization function into an on-disk format which is given to the host system to store through some system provided file writing function, stored by the host system, and can then be read through some system provided function, then deserialized and interpreted by some deserialization function. This series of actions describe the fundamental actions taken on file:

1. interface to modify content in working memory
2. transform from working memory to non-volatile memory or to stream
3. transfer to non-volatile state or by stream
4. transform from non-volatile state or from stream
5. process for use

It is in these steps that developer burden can be lessened. Lessening the difference between working memory and non-volatile memory in order to reduce the burden of processing input. Another is by lessening the difference of transferring to non-volatile forms and making data accessible over a network. These differences arise in part from encoding efficiency, endianness, and runtime data not relevant towards the model stored in saved files.

All in memory data structures can be practically represented by records (non-ordinal, possibly key-value), lists(ordinal), real numbers, integer numbers, booleans, null, and character strings. This has been noticed and designed around in SGML, JSON, TOML, YAML, and EDN. These formats are reliably interconvertible except for SGML because SGML allows representing identical information in semantically different ways.

Another more powerful design is available in the programming language Lisp, in particular the Scheme dialect. Given appropriate serializers and deserializers, this type of interface can provide a common approach for handling file data programmatically. By extending the concept of data serializers and deserializers to Lisp, a system for handling files which use a programmatic interface which can be directly supported by the OS. With more direct OS-like support, the host can act more intelligently on the data on behalf of a programmer to allow greater deduplication, transparent distribution, and greater integration with the host system.

2 Terms and Definitions

JSON:
File Sharding:
SGML:
EDN:
YAML:
Scheme:
OS:
Host:
Working Memory:
Serialization:
Deserialization:
TOML:
Non-volatile Memory:

LaTeX:
Library:
Grammar:
Recognizer:
S-expression:
Greenspun's Tenth Rule:
MongoDB:
Content Addressable Storage:
HPC:
Bit Torrent:
Object Storage:
Public Key:
Private Key:

3 Current File Problems and Solutions

The classical tradeoff when creating files is weighing space efficiency, encoding complexity, and understandability. Historically, the more ground up approach has led to a nightmare of backwards and forwards compatibility such as Microsoft's `.doc` format, and legacy `.rom` files which can just lose all meaning outside of their original systems. File type development and management is already a well known nightmare from several perspectives. Examples of managing this increased burden are: Apple's UTI, MIME, container formats, registering file extensions with a system, custom file identification programs like `file` or the United Kingdom's National Archive's Digital Preservation's `droid`, or other more esoteric and inconsistently applied identification schemes. This is to say little of the problem of determining what programs can even interact with a file. Adding structure with a programmatic interface, abstracting out serializers and deserializers, and adding a format recognition grammar should be easier.

4 Offering a Stronger Framework for Files

A more integrated, and more complete framework ostensibly involves more work, but, given the difficulty of working with current formats, should incur no extra burden. Hints at a simpler use case come from web development. Web development has moved towards JSON like serialization. Over custom approaches, using JSON simplifies interoperability, allows change over time, and lowerers developer effort.

In order to support general format recognition each program must implement a discrete input recognizer. Each recognizer should be registered with the host so that it can be applied to stored data and recognize a mapping of files to supported programs to be generated.

Recognizers have two steps levels. First, a grammar definition which can be used by a host system to preingest the file contents without executing the pro-

gram's code. It does this by having the grammar registered or programmatically available and then compiles or interprets the grammar. This allows guaranteed finite decidability for the grammar to remove any candidate programs and avoid all false negatives for determining what is compatible. Next, the full program can be invoked to recognize the file and terminate with a notice of recognition, unrecognition, or in the case of some other event non-recognition.

To support this kind of functionality in programs suggests a new executable format be used to contain these kinds of changes consistently. Currently, library support and some explicit action by programmers will be required. A later paper should be able to incorporate recognizer support more transparently, as well as addressing other shortcommings which has caused a large amount of infrastructure to be made to cope.

4.1 How to Represent Data

Beyond these mechanics, there are some important practices which should be observed in order to tightly represent and map observations to data. Often there is a raw number, state, or information observed and measured. A full context of the event must accompany each measure such that each measurement is meaningful and complete on its own. Optimizations should not be taken, but delegated to the serializer. Not only this, but files need to be complete, independant, and meaningful. A camera image, for example, then should record its specifications, ID, environmental conditions like temperature, self health tests, raw voltages from the sensor, and transformation algorithm and parameters. File may also contain a cache of the processed and transformed data as a convenience or as a way to reduce burden on downstream programs which may not support the code for the transformation.

5 Serialized Syntax

The serialized syntax is a meaningful description in order to convey the standard at a high level. Lisp was chosen as a syntax for this standard as it provides a format which is easy to formalize and is an efficient and readable form. LISP was also chosen in hopes of leapfrogging Greenspun's Tenth Rule of programming. JSON s-expressions are a manifest example of Greenspun's Tenth Rule. The following explain the grammar in Appendix A.

5.1 Core Representation

The core reserved top level key is `data`. This value is not intended to be accessed directly, but through a supported programming interface. This is because the actual form of the value associated with the `data` key can be in many different forms. Programs should only be concerned with the manifest content and not the literal form. A program will be given an abstract interface to access, possibly distributed data. Decrypting data is delegated tothe host by default.

This separation forms the entire crux of enabling greater security, extensibility, and compatibility. The value can be some serialized form of binary, a LISP data structure, partitioned into discrete chucks, refer to a number of other files for their data, encrypted, or something else entirely different. It is the responsibility of the host system, service, API, or library to resolve these and give a consistent view of the data.

Legacy formats may be included without more advanced features like automatic program compatibility recognition. Legacy descriptors are added in order to include information which could be inferred by using the format prescribed in this document. While full support cannot be given, support for more expansive descriptive keys, transparent distributed and sharded support, and enhanced security for distributing files can be given.

5.2 Transparent Distributed and Sharded Storage

To start, Bit Torrent, Object Storage, Content Addressable Storage, and Distributed Files systems have shown that there is an obvious need for distributed file storage and file replication. HPC environments need distributed data for work parallelization.

5.3 Support and use of Content Addressable Storage

In addition to data and security sections, support for Content Addressable Storage (CAS) is easily obtained by including a hash specification. Hashes of the content can be added after the fact or included at file creation. This is done by giving a list of fully qualified fields, what hash parameters were used, and the hashed value. These hashinging parameters and hash values can be used to match a file or search for a file for use in a CAS system in a way similar to MongoDB. Hashing must be done on the unencrypted values. Searching for a hash corresponding list of keys is an effective way to have robust and flexible searching for a CAS. The host can match against local files and transparently ask other hosts. For small files, preimage attacks should be taken into consideration.

5.4 Securing data

There is a need to have a general, extensible, and robust means of securing data in files. Keeping secure and private data in a public space is also important for future computing needs. One current use cases are to replicate local files on other systems safely for backup or archival purposes. Another is to distribute access controlled files on external networks in a fully distributed, sharded, global way while also keeping data access limited according to institutional standards. In order to accomplish this degree of secrecy, classical PKI is insufficient to match the flexibility needed. In order to do so the following is needed: A randomly generated key with 256 bits of entropy or greater which is used to encrypt all data with a symmetric cipher like AES-256. The key is then encrypted through a set of entities' private keys, remote services which take

user credentials, and if desired for a set of specific recipients, that recipient's public key. This can be performed a number of times with the key for each intended recipient. Such signing can be nested and applied selectively to fully qualified keys so that a limited set of encrypting entities are known at first to ensure greater privacy. Each one of these must contain sufficient information to identify the needed keys and approving entities to obtain the next level of signees until the key is obtained and data can be decrypted.

In some situations, as with the sharded file, the security scheme can be inferred. In this case, the top level key '`IS`' is specified to denote "inherit Security" in order to make the data overhead acceptable.

Support for using external servers to contact to decrypt a key has been lightly added and considered. While a simple incomplete case has been added to the grammar, this functionality is largely left to implementers. The rules to recursively define security have been added to give a fully flexible way to define arbitrary security structures of needed approvals or accessible keys to use the data. These also include a cannonical way to indicate which fields have their values encrypted and by what means or pathways they are encrypted so that mixed security can be implemented and enabled.

5.5 Data Integrity

Beyond usual security needs is the need for very long term reliable security. This can be accomplished by chaining trusted signatures which sign the data and the datetime the signature is being applied. Upon each transfer of the data, a new signature of existing signatures and datetime must be applied in order to maintain chain of custody. In the event a key is compromised or expires, so long as there is a chain of custody where each signature is signed before the time of of compromise or expiry the data is able to be verified and trusted to the same degree as the weakest signature in the strongest valid subset of signatures which are sufficient to establish a chain of custody. This chain grows linearly with the number of transfers, and reducing this is an open problem.

6 Application Programming Interface

Use and integration of the file features listed above must be automatic with no programmer overhead to allow the adoption of this new file standard. Listed file features should also be able to be controlled in a more fine grained mannar, but such usage should never be the default. Only the logical interfaces are specified. There is not a cannonical implementation when writing this document. There may never be a cannonical implementation, only a standard with other implementations.

6.1 Universal format components

in memory representation/API structure additional aspects incorporating outside grammars and vocabulary abstract file structure deserialization from any given format deserialization from the universal format serialization to any given format serialization»s« to the universal format

A Grammar

⟨*S*⟩ ::= '`(`' $unordered$ (⟨*DATA*⟩, ⟨*SECURITY_SPECIFICATION*⟩) '`)`'

⟨*DATA*⟩ ::= '`(`' '`data`' '`(`' '`legacy`' ⟨*BASE64_ENCODED_BINARY*⟩ '`)`' '`)`'
| '`(`' '`data`' '`(`' '`simple`' ⟨*LISP_STRUCTURED_DATA*⟩ '`)`' '`)`'
| '`(`' '`data`' '`(`' '`distributed`' ⟨*DATA_BLOCKS*⟩ '`)`' '`)`'

⟨*DATA_BLOCKS*⟩ ::= ⟨*HASH_RECORD*⟩ ⟨*DATA_BLOCKS*⟩
| ⟨*HASH_RECORD*⟩

⟨*HASH_RECORD*⟩ ::= '`(`' '`hash record`' '`(`' ⟨*KEY_FEILDS*⟩ '`)`' '`(`' ⟨*HASH_DETAILS*⟩ '`)`' ⟨*BENCODE_BINARY*⟩ '`)`'

⟨*SECURITY_SPECIFICATION*⟩ :: ⟨*SECURITY_FEILDS*⟩
| '`(`' '`IS`' '`)`'
| *null*

⟨*SECURITY_FEILDS*⟩ ::= ⟨*SECURITY_FEILD*⟩ ⟨*SECURITY_FEILDS*⟩
| ⟨*SECURITY_FEILD*⟩

⟨*SECURITY_FEILD*⟩ ::= ⟨*APPLY_PUBLIC_KEY_PAIR*⟩
| ⟨*APPLY_PRIVATE_KEY_PAIR*⟩
| ⟨*APPLY_SERVER_KEY*⟩
| ⟨*NEXT_DEPTH_SECURITY*⟩
| ⟨*LIST_ENCRYPTED_FEILDS*⟩

⟨*APPLY_PUBLIC_KEY_PAIR*⟩ ::= '`(`' '`apply_public_key`' ⟨*PKI_ID*⟩ '`)`'
⟨*PKI_ID*⟩ ::= ⟨*EMAIL_ADDRESS*⟩

⟨*APPLY_PRIVATE_KEY_PAIR*⟩ ::= '`(`' apply_private_key ⟨*PKI_ID*⟩ '`)`'

⟨*APPLY_SERVER_KEY*⟩ ::= '`(`' apply_server_key ⟨*SERVER_SCHEME*⟩ ⟨*URL*⟩ '`)`'

⟨*NEXT_DEPTH_SECURITY*⟩ ::= ‘`(`’ next_depth_security
⟨*BASE64_ENCODED_BINARY*⟩ ‘`)`’

⟨*LIST_ENCRYPTED_FEILDS*⟩ ::= ‘`(`’ ‘`encrypted_feilds`’ ‘`(`’
⟨*KEY_FEILDS*⟩ ‘`)`’ ‘`)`’

⟨*KEY_FEILDS*⟩ ::= ⟨*UTF8_STRING*⟩ ⟨*KEY_FEILDS*⟩
| ⟨*UTF8_STRING*⟩

⟨*LEGACY_SPECIFIER*⟩ ::= ‘`(`’ ‘`Legacy`’ BOOLEAN_STRING ‘`)`’

⟨*BOOLEAN_STRING*⟩ ::= ‘`true`’
| ‘`false`’

⟨*LEGACY_APPLICATIONS*⟩ ::= ‘`(`’ ‘`legacy compatible applications`’
⟨*URL_LIST*⟩ ‘`)`’

⟨*URL_LIST*⟩ ::= ‘`(`’ ⟨*URLS*⟩ ‘`)`’

⟨*URLS*⟩ ::= ⟨*URL*⟩ ⟨*URLS*⟩
| ⟨*URL*⟩