The Format Galaxy (DRAFT)

Felix Kohlgrüber | 2021-XX-XX

Text Editors

- Have been around for ~50 years
- Open file → Text appears on screen → Edit → Save
- Editing is natural and efficient
- Require text files
-

Text files

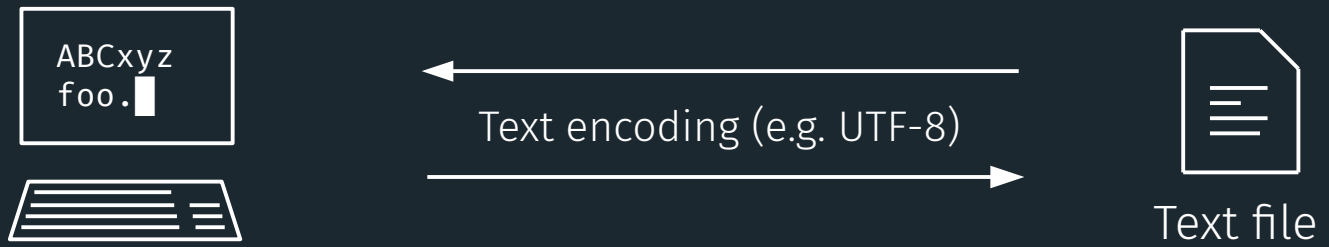
- Subset of binary files
- Sequence of characters


```
ASCII:
A ⟷ 0×41
B ⟷ 0×42
...
```

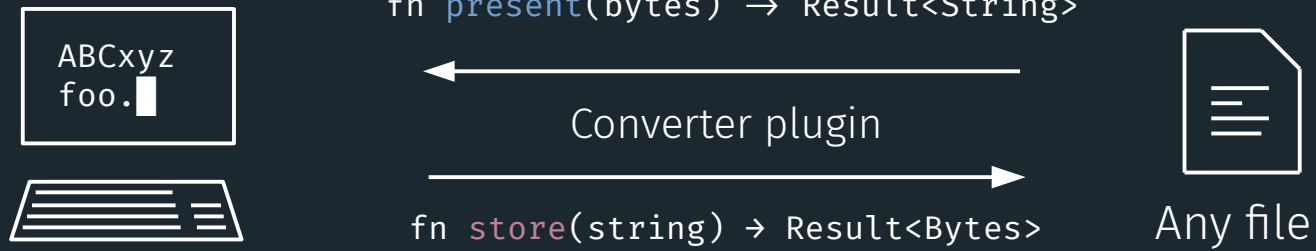
- Pros
  - Compatiblity
- Cons
  - Efficiency
  - Low-level format
  - Mix of content and presentation

The Idea

Regular text editor



Generalized text editor

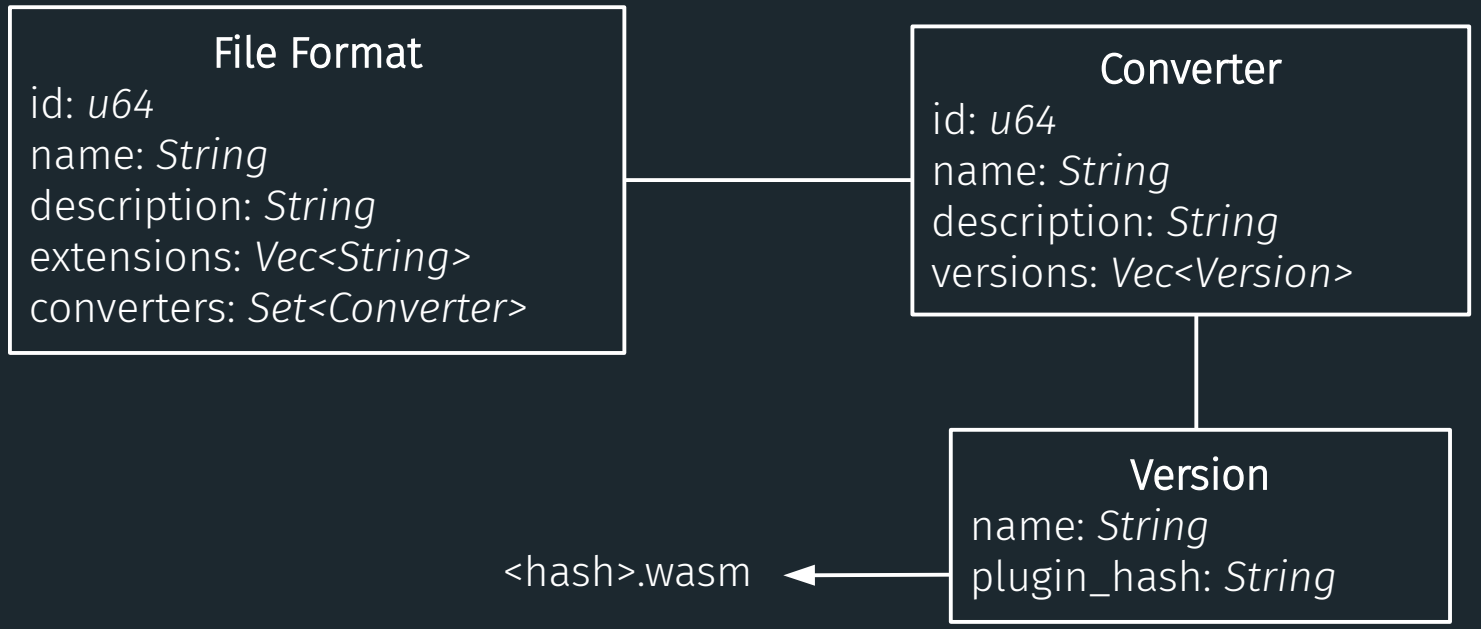


Converter Plugins

- WebAssembly modules (*.wasm file)
- Two functions:
  - `fn present(bytes: &[u8]) → Result<String, String>`
  - `fn store(s: &[str]) → Result<Vec<u8>, String>`
- Workflow
  - User opens some file
  - Editor checks file type and lets user choose a compatible converter plugin
  - Plugin's `present` is used to convert the file content into a text
  - Regular text editing by user
  - When saving, plugin's `store` is used to convert edited text into storage representation

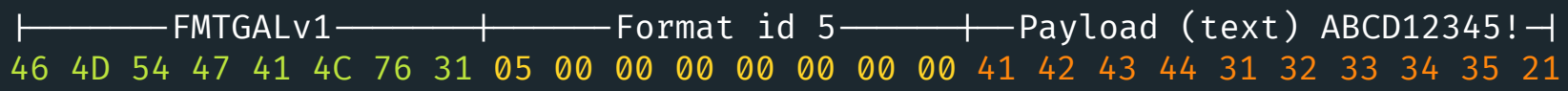
Format Galaxy Index

- Public registry of converter plugins



Format Galaxy Container Format

- Simple wrapper format storing the format id of the file's content
- File extension: `.fg`
- Format: `<prelude><format_id><payload>`
  - `prelude`: "FMTGALv1" encoded in ascii
  - `format_id`: 64-bit little-endian unsigned integer identifying the format
  - `payload`: actual data to be stored
- Example:



Json-Like Format

- Binary format having a data model similar to JSON
- Type-Length-Value (TLV) encoding
  - `type`: single byte encoding the type of value (bool, number, ...)
  - `length`: 64-bit LE unsigned integer encoding the length / number of elements (for some variants)
  - `value`: actual data to be stored for each variant

```
pub enum Value {
    Null,
    Bool(bool),
    Number(u64),
    String(String),
    Array(Vec<Value>),
    Object(IndexMap<String, Value>),
}
```

```
type length value
0
1           1 byte boolean (0=false, 1=true)
2           8 byte LE unsigned integer
3    <len>  <len> bytes utf-8 encoded string
4    <len>  <len> Values
5    <len>  <len> pairs of a string (length+value)
            and a Value
```

Json-Like Format - Examples

```rust
pub enum Value {
    Null,
    Bool(bool),
    Number(u64),
    String(String),
    Array(Vec<Value>),
    Object(IndexMap<String, Value>),
}
```

```
type length value
0
1           1 byte boolean (0=false, 1=true)
2           8 byte LE unsigned integer
3    <len>  <len> bytes utf-8 encoded string
4    <len>  <len> Values
5    <len>  <len> pairs of a string (length+value)
            and a Value
```

- Examples:

```
- 00                                     Null
- 01 00                                  Bool(false)
- 01 01                                  Bool(true)
- 02 09 00 00 00 00 00 00 00             Number(9)
- 03 03 00 00 00 00 00 00 00 61 62 63    String("abc")
- 04 02 00 00 00 00 00 00 00 01 01 00    Array(Bool(true), Null)
- 05 02 00 00 00 00 00 00 00
  01 00 00 00 00 00 00 00 61 01 01
  01 00 00 00 00 00 00 00 61 00          Object("a": Bool(true), "b": Null)
```