Подходы к созданию проприетарного формата представления данных

Г.М. Кряхтунов, А.С. Боронников

***Аннотация* — При разработке программного обеспечения неизбежно возникает вопрос о выборе формата представления данных, которые определяют, как информация будет организована для эффективного хранения, передачи и обработки данных. Однако известные универсальные форматы не всегда подходят по конкретным требованиям и критериям, поэтому возникает необходимость в разработке собственного.**

В данной статье рассматриваются вопросы создания схемы представления данных для разработки собственных проприетарных форматов представления данных.

В работе проведен анализ основных метрик форматов хранения данных, в результате которого предложены критерии к схеме данных. Выделены основные преимущества форматов хранения данных в ходе краткого обзора существующих форматов. На основе полученной информации разработана схема данных для проприетарного формата представления данных, которая реализует иерархическую структуру с предложенными классами. Определена их структура, общий формат записи файла, типы записи и кодировка. Кроме этого, схема данных обеспечивает контроль целостности информации путем добавления контрольной суммы CRC в конец файла.

В качестве примера применения разработанной схемы данных, в работе представлено создание закрытого проприетарного формата LSGS (Library of symbolic graphic symbols), предназначенного для хранения данных библиотеки условных графических обозначений функциональных блоков. Представлен результат сериализации и десериализации объектов на языке программирования высокого уровня.

***Ключевые слова* — формат хранения данных, схема представления данных, схема данных, формат, data storage format, relationships, data patterns, LSGS, JSON, XML, CSV, HDF5, AVRO, Intel HEX, ORC.**

I. Введение

При разработке различного программного обеспечения (далее – ПО) на одном из этапов встает вопрос об организации информации в файлах, а именно в выборе формата представления данных (далее – формат). Для решения этой задачи выдвигается ряд требований, соответствующих определенной предметной области. Форматы обычно определяют, как информация будет закодирована и организована для эффективного хранения, передачи и обработки данных. Они используются для хранения различных типов данных. Зачастую универсальные форматы не подходят по одному или нескольким критериям, поэтому возникает необходимость в разработке собственного.

В данной статье предлагаются подходы к созданию схемы представления данных (далее – схема данных), на основе которой разрабатывается закрытый проприетарный формат для объединения функциональных элементов в составе одной библиотеки.

Статья получена 29 января 2024.

Глеб Михайлович Кряхтунов, Кафедра вычислительной техники, МИРЭА – Российский технологический университет, Москва, Россия (e-mail: gleb.kryakhtunov@yandex.ru).

Антон Сергеевич Боронников, Кафедра вычислительной техники, МИРЭА – Российский технологический университет, Москва, Россия (e-mail: boronnikov-anton@mail.ru).

II. Основные метрики форматов

Проприетарным форматом является формат, не являющийся открытым стандартом. Его обычно применяют с целью обеспечения полного контроля и конфиденциальности данных внутри организации или проекта. Большинство проприетарных форматов являются *закрытыми*, т.е. доступ к внутренней структуре файла предоставляется только владельцам или разработчикам, ответственным за создание и поддержку формата. Иногда владельцы могут предоставлять ограниченный доступ к спецификациям формата через лицензирование или другие соглашения, что может допускать определенные уровни взаимодействия с данными внешними сторонами. Использование такого вида формата связано с некоторыми практическими соображениями, такими как обеспечение полного контроля над тем, как данные структурируются, кодируются и обрабатываются, а также конфиденциальность данных.

Не менее важным требованием к формату является поддержка групп объектов и метаданных. Группы объектов представляют собой механизм, позволяющий объединять связанные элементы данных в структурированные наборы. Это особенно полезно при работе с большими объемами информации или сложными иерархическими данными. Метаданные необходимы для добавления сведений об объектах, таких как информация об авторе, дата создания, версия и т.д. Простыми словами метаданные – это свойства объекта. Эти требования обеспечивают не только эффективную организацию данных, но и улучшают их понимание и масштабируемость, делая формат более гибким. Методы идентификации базовых и сложных типов объектов устанавливает разработчик формата.

Кодировка содержимого файла определяет стандарт кодирования символов внутри файла. Среди множества различных видов кодировок наиболее используемые – это ASCII [1], UTF-8 [2] и Windows-1251 [3]. Выбор кодировки зависит от требований конкретного

приложения и поддержки языковых символов. Рекомендуется при сохранении файла с данными использовать кодировку UTF-8 по следующим причинам:

- универсальность: UTF-8 является универсальной кодировкой, позволяющей представлять символы практически всех письменных языков мира. Это делает ее идеальным выбором для форматов данных, которые могут содержать информацию на различных языках;
- эффективность хранения: UTF-8 обеспечивает эффективное использование памяти и хранения данных, поскольку представляет большинство символов одним байтом и расширяется до многобайтовых представлений, не входящих в базовый набор символов кодировки ASCII. Это особенно важно при работе с текстом на языках, использующих кодировку ASCII;
- совместимость: UTF-8 легко обрабатывается и поддерживается многими современными программами и платформами. Это обеспечивает хорошую совместимость и обмен данных между различными системами и приложениями.

Проблема *обратной совместимости* с более ранними или поздними версиями ПО в форматах заключается в том, как обеспечить корректное взаимодействие между различными версиями программ, способных читать и записывать данные согласно заданному формату. Эта проблема может возникнуть из-за изменений структуры, синтаксиса или смысловой значимости элементов формата между разными версиями программ. Для решения этой проблемы часто внедряют систему *версионирования* формата, позволяющую ПО более ранней версии не обрабатывать структуру и синтаксис более позднего формата или распознавать ту структуру и данные, которые соответствуют более ранней версии формата.

Формат может содержать средства *контроля целостности данных*, которые позволяют проверить содержимое файла на изменение информации. В качестве таких средств обычно выступают следующие методы – использование контрольных сумм CRC [4] и хеш-функций [5], применение цифровых подписей [6], а также шифрование данных. Для обеспечения целостности данных в проприетарных закрытых форматах обычно используется контрольная сумма CRC (Cyclic Redundancy Check) – некоторое значение, рассчитанное по набору данных путем применения табличного метода расчета, либо с помощью побитового сдвига. CRC хорошо подходит для обнаружения случайных ошибок, таких как непредвиденные битовые инверсии или помехи в процессе передачи данных. Преимуществом данного подхода является легкая реализация и набор варьируемых параметров, используемых при расчете контрольной суммы. Однако следует учесть, что контрольной суммы может быть недостаточно для защиты целостности данных, так как ее можно обойти при злонамеренных манипуляциях с данными, поэтому для проприетарных форматов в закрытом ПО, желательно рассмотреть использование более сложных методов, таких как криптографические хеш-функции и цифровые подписи, чтобы обеспечить более высокий уровень защиты информации. Но где риски повреждения или изменения данных невысоки, контрольной суммы CRC достаточно для обеспечения целостности.

Таким образом, в результате анализа основных метрик форматов были предложены критерии схемы представления данных для разработки проприетарного формата (рис. 1).

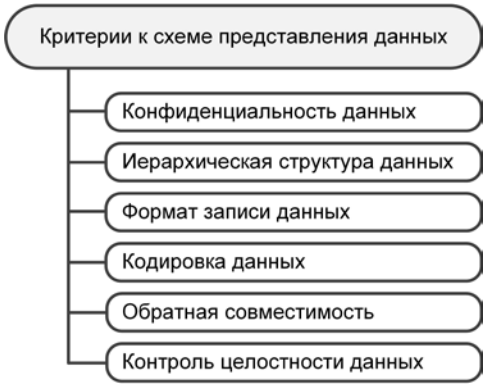

Рис. 1. Критерии к схеме представления данных

III. Обзор форматов

Процесс преобразования структуры данных, семантически понятной человеку, в последовательность битов для машинной обработки называется *сериализацией*. Характер этой операции преобразования объекта в поток байтов для сохранения и передачи в память, базу данных или файл определяется *схемой представления данных*. Для знакомства с используемыми схемами представления данных и видами сериализованных объектов ниже были рассмотрены основные форматы хранения данных.

JSON (JavaScript Object Notation) – формат, основанный на языке JavaScript. Он основан на двух структурах данных – коллекция пар ключ/значение и упорядоченный список значений. JSON не включает встроенных средств контроля целостности данных [7]. Формат позволяет сохранять такие структуры как словари и списки в структурированном формате [8]. JSON поддерживает такие типы и структуры как строка, число, логический тип, массивы, значение null и сложные объекты [9]. Пример содержимого файла JSON изображен на рис. 2.

```
{
  "users": [
    {
      "id": 1,
      "name": "Gleb Kryakhtunov",
      "age": 22,
      "email": "gleb.kryakhtunov@yandex.ru"
    },
    {
      "id": 2,
      "name": "Anton Boronnikov",
      "age": 27,
      "email": "boronnikov-anton@mail.ru"
    }
  ]
}
```

Рис. 2. Пример содержимого файла JSON

XML (eXtensible Markup Language) — расширяемый язык разметки. XML универсален и подходит для описания различных данных. Структура XML-документа определена набором элементов, который включает в себя теги и атрибуты. Теги в документах всегда идут в виде пар, что обеспечивает легкость восприятия данных. Возможность вложенности тегов и использование атрибутов позволяют представлять данные в виде дерева тегов [10]. XML, как и JSON, не включает встроенных средств контроля целостности, но для этого можно дополнительно использовать схемы XSD [11]. Пример содержимого файла XML изображен на рис. 3.

```
<users>
  <user>
    <id>1</id>
    <name>Gleb Kryakhtunov</name>
    <age>22</age>
    <email>gleb.kryakhtunov@yandex.ru</email>
  </user>
  <user>
    <id>2</id>
    <name>Anton Boronnikov</name>
    <age>27</age>
    <email>boronnikov-anton@mail.ru</email>
  </user>
</users>
```

Рис. 3. Пример содержимого файла XML

CSV (Comma-Separated Values) представляет собой текстовый формат для представления табличных данных, где каждая строка файла соответствует строке таблицы. Несмотря на название файла, разделителем колонок может быть не только символ запятой. Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками [12]. Пример содержимого файла CSV изображен на рис. 4.

```
id,name,age,email
1,Gleb Kryakhtunov,22,gleb.kryakhtunov@yandex.ru
2,Anton Boronnikov,27,boronnikov-anton@mail.ru
```

Рис. 4. Пример содержимого файла CSV

Формат HDF5 (Hierarchical Data Format version 5) представляет открытый стандарт для хранения и управления большими данными Big Data. Его иерархическая структура подобна принципу файловой системы, где для доступа к данным используются пути, аналогичные POSIX-синтаксису (например, /path/to/resource). Формат включает два основных класса объектов: набор данных (Datasets), представляющих собой многомерные массивы объектов одного типа, и группы (Groups), которые выступают в качестве контейнеров для наборов данных и других групп. Метаданные хранятся в виде набора именованных атрибутов объектов. Способность шифрования и сжатия данных обеспечивает безопасность и эффективность хранения [13]. Пример содержимого файла HDF5 для наглядности изображен в текстовой форме на рис. 5, но обычно данный формат используется для бинарного хранения данных в виде набора нулей и единиц.

```
/HDF5/users
/HDF5/users/user_1
/HDF5/users/user_1/id     1
/HDF5/users/user_1/name   "Gleb Kryakhtunov"
/HDF5/users/user_1/age    22
/HDF5/users/user_1/email  "gleb.kryakhtunov@yandex.ru"
/HDF5/users/user_2
/HDF5/users/user_2/id     2
/HDF5/users/user_2/name   "Anton Boronnikov"
/HDF5/users/user_2/age    27
/HDF5/users/user_2/email  "boronnikov-anton@mail.ru"
```

Рис. 5. Пример содержимого файла HDF5

Наибольшую популярность среди применяемых форматов в Big Data занимает формат AVRO – линейно-ориентированный (строчный) формат хранения файлов. Формат сохраняет схему данных в независимом от реализации текстовом формате JSON. AVRO представляет собой контейнер, содержащий в себе заголовок и один или несколько блоков с данными (рис. 6). Заголовок состоит из:

- ASCII слова «Obj1»;
- Метаданных файла, включая определение схемы;
- 16-байтного случайного слова (маркера синхронизации).

Блоки данных в свою очередь состоят из:

- Количества объектов в блоке;
- Размера сериализованных объектов в блоке, определяющимся в байтах;
- Данных объекта, представленных в двоичном виде с конкретным типом сжатия;
- Маркера синхронизации.

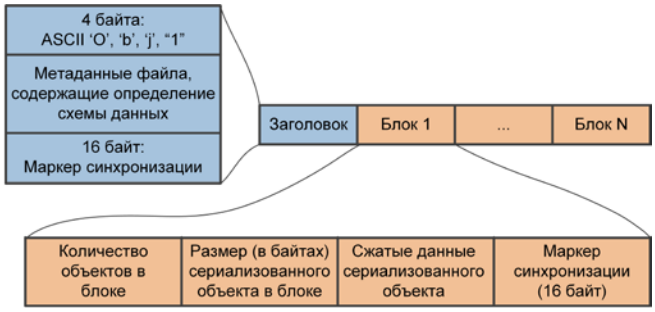

Рис. 6. Структура файла AVRO

Для блоков данных AVRO можно использовать компактную бинарную кодировку или формат JSON. AVRO также поддерживает эволюцию схем данных, обрабатывая изменения схемы путем пропуска, добавления или модификации отдельных полей записи. AVRO не является строго типизированным форматом: информация о типе каждого поля хранится в разделе метаданных вместе со схемой данных. Благодаря этому для чтения сериализованной информации не требуется предварительное знание схемы данных. AVRO поддерживает многие типы данных [14]. Пример содержимого файла AVRO изображен на рис. 7, где вначале представлена схема данных, а затем набор записей согласно ей.

```
{
  "type": "record",
  "name": "User",
  "fields": [
    {"name": "id", "type": "int"},
    {"name": "name", "type": "string"},
    {"name": "age", "type": "int"},
    {"name": "email", "type": "string"}
  ]
}

[
  {
    "id": 1,
    "name": "Gleb Kryakhtunov",
    "age": 22,
    "email": "gleb.kryakhtunov@yandex.ru"
  },
  {
    "id": 2,
    "name": "Anton Boronnikov",
    "age": 27,
    "email": "boronnikov-anton@mail.ru"
  }
]
```

Рис. 7. Пример содержимого файла AVRO

Intel HEX – это текстовый формат, используемый для представления двоичных данных. Каждый байт данных представлен в виде пары символов в кодировке ASCII. Запись формата включает такую информацию, как маркер записи, количество данных, смещение, тип записи, данные, контрольная сумма. Каждая строка в файле Intel HEX содержит одну запись данных, указывающую расположение данных в памяти. В настоящее время определены различные типы записей организующих хранение данных с помощью линейной адресации, адресации по сегментам, расширенной линейной адресации [15]. Общий формат записи изображен на рис. 8.

Маркер записи	Кол-во данных	Смещение	Тип записи	Данные	CRC
:	1 байт	2 байта	1 байт	N байт	1 байт

Рис. 8. Общий формат записи Intel HEX

Этот формат обычно используется для загрузки программ в память микроконтроллеров и других встраиваемых устройств с помощью специализированного оборудования (например, программатор). Пример содержимого файла формата Intel HEX изображен на рис. 9.

```
:10000000BE6E97A9F36A0BBB0037D19F0C7844EE04
:1001000002C65FE922E941E0AA2D294817CB54FD29
:10002000E4BF68168601A9418D0374582ED0E6CD31
:10003000F735E51878091AD52CB7FFC640C2A434A5
:10004000E2D3BA4C0F2E67011D1D5B1E74E8B1D9B7
:00000001FF
```

Маркер записи | Тип записи | Количество данных | Данные | Смещение | Контрольная сумма

Рис. 9. Пример содержимого файла Intel HEX

Все рассмотренные выше форматы являются *строковыми*, так как они хранят данные в виде строк. Существует иной тип форматирования данных, который относится к группе *колоночных* форматов. Наиболее распространенным из них является формат ORC (Optimized Record Columnar File). ORC оптимизирован для чтения потоков Big Data, а также включает интегрированную поддержку быстрого поиска нужных строк данных. Формат ORC осуществляет хранение набора строк данных в виде столбцов. ORC-файл разделяет исходный набор данных на полосы (stripes, страйпы) объемом по 250 МБайт. Колонки в таких страйпах разделены друг от друга, и хранят в себе индексы, данные и метаданные. В конце файла записываются параметры сжатия, размер блока данных и другая необходимая информация (postscript) [16]. На рис. 10 приведена схема структуры формата ORC.

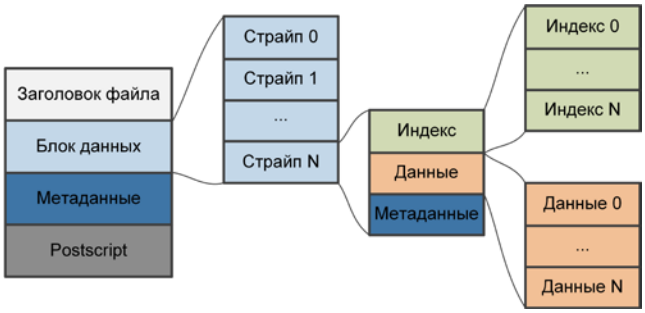

Рис. 10. Структура формата записи Intel HEX

На основе анализа рассмотренных структур были выделены преимущества для разработки схемы представления данных формата:

- Для описания структуры объекта или группы сложных объектов подходят такие форматы, как JSON, XML, HDF5;
- JSON, XML и HDF5 хорошо подходят для описания иерархических структур данных, а формат CSV для табличных;
- AVRO позволяет описать данные любой сложности. Преимуществом является поддержка изменения схемы представления данных в файле путем пропуска, добавления или модификации отдельных полей записи;
- Intel HEX позволяет описать двоичные данные с помощью кодировки ASCII. Главными преимуществами формата Intel HEX является то, что каждая запись закрывается контрольной суммой CRC для обеспечения целостности данных и каждый байт кодируется в виде пары шестнадцатеричных ASCII-символов;
- Рассмотренные строковые форматы описываются с помощью набора записей, которые разделяются с помощью различных спецсимволов;
- У форматов AVRO, Intel HEX и ORC содержатся метаданные для идентификации полей и структуры записей.

IV. Описание схемы представления данных

На основе предложенных критериев и преимуществ рассмотренных решений предлагается схема данных для разработки собственного проприетарного формата, который обеспечивает хранение простой коллекции объектов и данных.

Схема данных реализует иерархическую структуру с двумя классами:

– Файл (рис. 11);
– Объект (рис. 12).

Объект – это набор свойств. Свойства определяются различными типами данных, который устанавливает разработчик (например, числа, строки и т.д.). Количество свойств и объектов, а также уровней вложенности объектов устанавливает разработчик формата.

Описание содержимого файла закрывается контрольной суммой CRC. Она рассчитывается от всех записей файла. Параметры алгоритма для расчета CRC задаются разработчиком.

Порядок свойств и объектов является необязательным условием, кроме записи CRC – она должна идти в конце содержимого файла.

Рис. 11. Структура файла

Рис. 12. Структура объекта

Общий формат записи состоит из метаданных, данных и разделителя записи (рис. 13). Каждая запись разделяется символами возврата каретки CR (перемещение курсора в начало строки без перехода на новую) и перехода на новую строку LF.

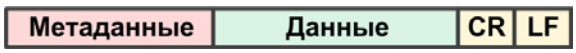

Рис. 13. Общий формат записи файла

Записи файла кодируются с помощью символов шестнадцатеричной системы счисления (0-9, A-F), где каждая пара символов отвечает за символ в кодировке UTF-8. Метаданные могут включать следующие поля:

– идентификатор (ID) программы;
– тип записи;
– тип данных;
– идентификатор (ID);
– количество байтов.

По схеме данных определено три типа записи – свойство, объект, контрольная сумма CRC. Форматы этих типов представлены на рис. 14-16.

Идентификатор программы предназначен для определения принадлежности записи файла к определенному ПО. В случае, если ПО имеет разные версии, то данный идентификатор может отличаться, тогда компилятор формата однозначно определяет к какой версии ПО относится данная запись. Таким образом, идентификатор программы обеспечивает обратную совместимость с более ранними версиями ПО.

					Данные в UTF-8				
ID прог.	Тип записи «0»	Тип данных	ID	Кол-во байтов N	Байт 1	...	Байт N	CR	LF
8 симв.	1 симв.	2 симв.	4 симв.	16 симв.	2*N симв.				

Рис. 14. Формат записи свойства

			Данные CRC				
ID прог.	Тип записи «F»	Кол-во байтов N	Байт 1	...	Байт N	CR	LF
8 симв.	1 симв.	2 симв.	2*N симв.				

Рис. 15. Формат записи CRC

				Кол-во свойств и объектов				
ID прог.	Тип записи «1»	ID	Кол-во байтов «8»	Байт 1	...	Байт 8	CR	LF
8 симв.	1 симв.	4 симв.	16 симв.	16 симв.				

Рис. 16. Формат записи объекта

Для кодирования типа записи достаточно одного символа, который задается разработчиком. В качестве примера тип записи свойства кодируется как «0», объект – «1», CRC – «F».

Поле «Тип данных» используется только в записи свойства. В качестве типов данных могут выступать базовые (int, float, char, long и т.д) или спроектированные разработчиком (дата и время, RGB-цвет и т.д.).

В метаданных свойства и объекта прописываются их идентификаторы для определения нужного свойства или объекта куда происходит запись данных.

Количество байтов в метаданных зависит от хранящихся данных. Например, количество байтов в формате записи CRC зависит от разрядности используемого полинома. Для CRC предусмотрено всего 2 символа, так как обычно используется CRC-8, CRC-16, CRC-32 и CRC-64, для описания данных которых необходимо 1 (0x01), 2 (0x02), 4 (0x04) и 8 (0x08) байт соответственно. Но существует также CRC-256, для описания которого необходимо 32 байта (0x10), поэтому 2 символа позволяют описать различные CRC с запасом. Для объектов количество байт устанавливается фиксированным значением “8”, так как с помощью 8 байт можно описать максимальное число объектов, что соответствует типу ulong во многих языках программирования.

Таким образом, схема представления данных позволяет объединять свойства и объекты по смысловой нагрузке, тем самым формируя универсальную

структуру, которую можно применить под определенную задачу.

V. Описание формата LSGS

В качестве примера рассмотрено создание проприетарного закрытого формата LSGS (Library of symbolic graphic symbols), предназначенного для хранения данных библиотеки условных графических обозначений функциональных блоков (далее – библиотека УГО ФБ). Данный формат будет применяться в реальной системе автоматизированного проектирования (далее – САПР).

При разработке формата выдвигается ряд требований:

- определенный набор типов хранимых данных;
- кодировка;
- безопасность.

LSGS должен поддерживать хранение следующей информации:

- Версия ПО;
- Название библиотеки;
- Версии библиотеки;
- Дата и время создания библиотеки;
- Дата и время изменения библиотеки;
- Автор (Разработчик) библиотеки;
- Описание библиотеки;
- Контрольная сумма CRC.

Объект версии библиотеки УГО ФБ должен содержать в себе:

- Список категорий УГО ФБ;
- Список УГО ФБ.

Список категорий УГО ФБ хранит информацию о категориях элементов УГО ФБ, которые включают в себя идентификатор и название категории.

Список УГО ФБ должен хранить информацию об элементах УГО ФБ, которые включают в себя:

- Название элемента;
- Идентификатор категории;
- Дата и время создания;
- Дата и время изменения;
- Автор (Разработчик);
- Описание;
- Путь к файлу УГО ФБ.

Формат должен осуществлять проверку содержимого файла на внешние изменения информации. Записи формата должны записываться в кодировке UTF-8. Благодаря поддержке идентификатора ПО, файл может содержать различные записи с одинаковой смысловой нагрузкой для разных версий ПО.

Исходя из поставленных требований разработанная схема данных подходит по всем параметрам для формата LSGS. Путем применения этой схемы данных формат имеет иерархическую структуру, которая включает в себя следующие уровни вложенности:

- Верхний уровень (рис. 17);
- Уровень версии (рис. 18);
- Уровень списка категорий УГО ФБ (рис. 19);
- Уровень списка УГО ФБ (рис. 20);
- Уровень категории (рис. 21);
- Уровень УГО ФБ (рис. 22).

Рис. 17. Верхний уровень формата LSGS

Рис. 18. Уровень версии формата LSGS

Рис. 19. Уровень списка категорий УГО ФБ формата LSGS

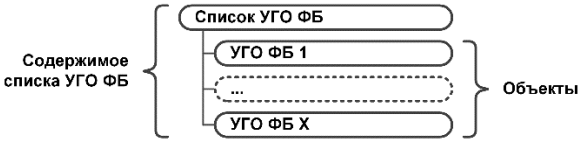

Рис. 20. Уровень списка УГО ФБ формата LSGS

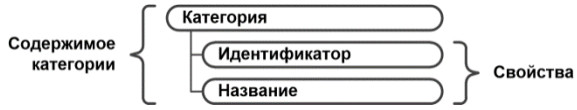

Рис. 21. Уровень категории формата LSGS

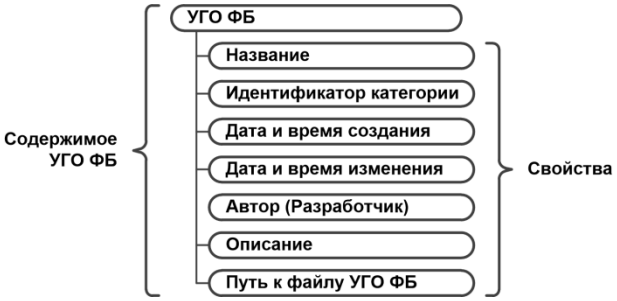

Рис. 22. Уровень УГО ФБ формата LSGS

Формат содержит три типа свойств:

- Строка (рис. 23);
- Дата и время (рис. 24);
- Целочисленное 32-разрядное число (рис. 25).

Свойства, относящиеся к строковому типу данных:

- Название библиотеки;
- Автор (Разработчик) библиотеки;
- Описание библиотеки;
- Название категории;
- Название УГО ФБ;
- Автор (Разработчик) УГО ФБ;
- Описание УГО ФБ;
- Путь к файлу УГО ФБ.

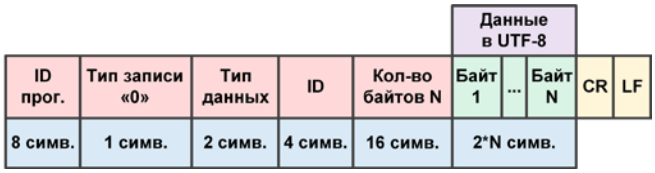

Рис. 23. Общий формат записи строки LSGS

Свойства, относящиеся к целочисленному типу данных:

- Идентификатор версии;
- Идентификатор категории;
- Идентификатор категории УГО ФБ.

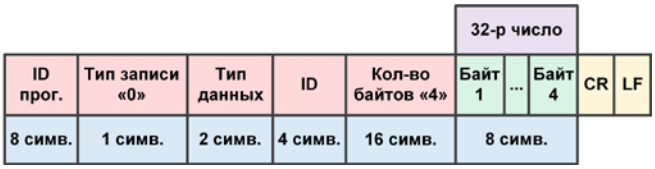

Рис. 24. Общий формат записи числа LSGS

Данные записи содержат 7 байтов данных, отвечающих за описание даты и времени. Первый байт отвечает за число месяца, второй – номер месяца, третий и четвертый – год, пятый – час, шестой – минуты, седьмой – секунды. Свойства, относящиеся к типу даты и времени:

- Дата и время создания библиотеки УГО ФБ;
- Дата и время изменения библиотеки УГО ФБ;
- Дата и время создания УГО ФБ;
- Дата и время изменения УГО ФБ.

Рис. 25. Общий формат записи даты и времени LSGS

Формат записи объектов соответствуют общему формату записи (рис. 16), где отличаются только идентификаторы объектов. Формат содержит пять типов объектов:

- Список версий библиотеки УГО ФБ;
- Версия библиотеки УГО ФБ;
- Список категорий УГО ФБ;
- Список УГО ФБ;
- Категория УГО ФБ;
- УГО ФБ.

Для контроля целостности данных используется CRC-32, тогда формат записи принимает следующий вид, представленный на рис. 26.

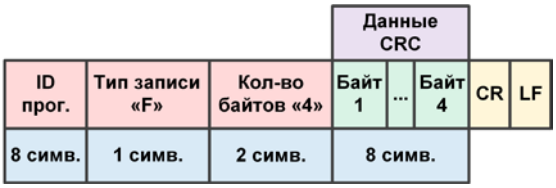

Рис. 26. Общий формат записи CRC LSGS

VI. Сериализация и десериализация формата LSGS

Так как LSGS разработан для коммерческого продукта, поэтому для примера были взяты случайные значения типов данных (табл. 1) и идентификаторов (табл. 2), а значение идентификатора программы принято за «00000000».

Таблица 1. Кодировка типов данных

Тип данных	Кодировка
Строка	0x00
Дата и время	0x01
Целочисленное 32-разрядное число	0x02

Таблица 2. Кодировка идентификаторов

Тип данных	Кодировка
Название библиотеки	0x0000
Дата и время создания библиотеки	0x0001
Дата и время изменения библиотеки	0x0002
Автор (Разработчик) библиотеки	0x0003
Описание библиотеки	0x0004
Список версий библиотеки	0x0005
Версия библиотеки	0x0006
Идентификатор версии библиотеки	0x0007
Список категорий	0x0008
Категория	0x0009
Идентификатор категории	0x000A
Название категории	0x000B
Список УГО ФБ	0x000C
УГО ФБ	0x000D
Название УГО ФБ	0x000E
Идентификатор категории УГО ФБ	0x000F
Дата и время создания УГО ФБ	0x0010
Дата и время изменения УГО ФБ	0x0011
Автор (Разработчик) УГО ФБ	0x0012
Описание УГО ФБ	0x0013
Путь к файлу УГО ФБ	0x0014

В качестве начального значения CRC-32 взято значение 0x00000000, а качестве полинома многочлен, представленный в формуле (1), который можно представить в виде битовой последовательности в шестнадцатеричной форме – 0x1543417F (не учитывая старшую степень полинома).

$$P(x) = x^{32} + x^{28} + x^{26} + x^{24} + x^{22} + x^{17} + x^{16} + \\ + x^{14} + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + x^0 \quad (1)$$

Диаграмма классов в нотации UML [17] изображена на рис. 27. На ней представлены следующие классы:

- библиотеки УГО ФБ (Library);
- версии (Version);
- категории (Category);

– элемент УГО ФБ (Element).

Каждый класс содержит два метода Serialize и Deserialize, отвечающих за сериализацию и десериализацию объектов данной структуры. В качестве примера были взяты данные библиотеки УГО ФБ, представленных на рис. 28. Для того чтобы сериализовать библиотеку УГО ФБ необходимо вызвать метод Serialize, которые сохранит данные библиотеки в файл формата LSGS (рис. 29). Метод Deserialize отвечает за обратную операцию – преобразует данные формата LSGS в объекты, описанные на языке программирования C#. Пример операции десериализации представлен на рис. 30.

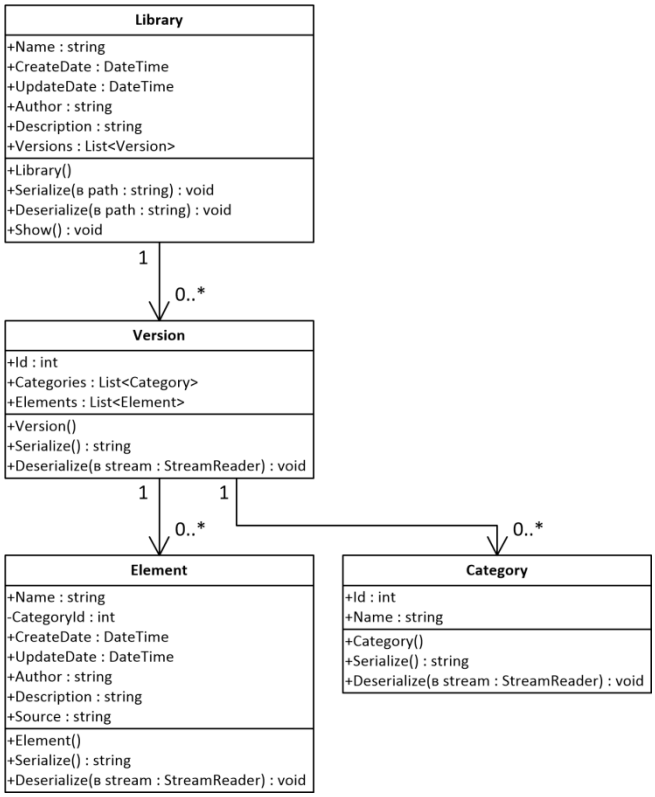

Рис. 27. Диаграмма классов

```
Name: Тестовая библиотека
CreateTime: 2024.01.28 04:36:36
UpdateTime: 2024.01.28 04:36:36
Author: Anton Boronnikov, boronnikov-anton@mail.ru
Description: Библиотека для отладки разработанного формата
Versions:
 Id: 1 Categories:
  Id: 1
  Name: Категория 1
 Elements:
  Name: УГО ФБ 1
  CategoryId: 1
  CreateTime: 2024.01.28 04:36:36
  UpdateTime: 2024.01.28 04:36:36
  Author: Anton Boronnikov, boronnikov-anton@mail.ru
  Description: УГО ФБ 1
  Source: Путь 1

 Id: 1 Categories:
  Id: 1
  Name: Категория 1
 Elements:
  Name: УГО ФБ 1
  CategoryId: 1
  CreateTime: 2024.01.28 04:36:36
  UpdateTime: 2024.01.28 04:36:36
  Author: Gleb Kryakhtunov, gleb.kryakhtunov@yandex.ru
  Description: УГО ФБ 1
  Source: Путь 1
```

Рис. 28. Данные библиотеки УГО ФБ

```
0000000000000000000000000000025D0A2D0B5D181D182D0BED0B2D0B0D18F20D0B1D0B8D0B1D
0BBD0E8D0BED182D0B5D0BAD080
0000000000100010000000000000071C0107E8033A08
0000000000100020000000000000071C0107E8033A08
0000000000000030000000000002A416E746F6E20426F726F6E6E696B6F762C20626F726F6E6
E696B6F762D616E746F6E406D61696C2E7275
0000000000000040000000000056D091D0B8D0B1D0B8D0B8D0BED182D0B5D0BAD0B020D0B4D
0BBD18F20D0BED182D0BBD0B0D0B4D0BAD0B820D180D0B0D0B7D180D0B0D0B1D0BED182D0B0D0B
DD0BDD08ED0B3D0BE20D184D0BED180D0BCD0B0D182D0B0
0000000010005000000000000008000000000000002
0000000010006000000000000008000000000000003
0000000002000700000000000000400000001
0000000010008000000000000008000000000000001
0000000010009000000000000008000000000000002
0000000002000A0000000000000400000001
000000000000000B000000000000014D09AD0B0D182D0B5D0B3D0BED180D0B8D18F2031
000000001000C000000000000008000000000000001
000000001000D000000000000008000000000000007
000000000000000E000000000000000DD0A3D093D09E20D0A4D0912031
0000000003000F000000000000000400000001
0000000000100110000000000000071C0107E8033A08
0000000000100120000000000002A416E746F6E20426F726F6E6E696B6F762C20626F726F6E6
E696B6F762D616E746F6E406D61696C2E7275
0000000000000130000000000000DD0A3D093D09E20D0A4D0912031
0000000000000140000000000000AD09FD183D182D18C2031
0000000010006000000000000008000000000000003
0000000002000700000000000000400000001
0000000010008000000000000008000000000000001
0000000010009000000000000008000000000000002
0000000002000A0000000000000400000001
000000000000000B000000000000014D09AD0B0D182D0B5D0B3D0BED180D0B8D18F2031
000000001000C000000000000008000000000000001
000000001000D000000000000008000000000000007
000000000000000E000000000000000DD0A3D093D09E20D0A4D0912031
0000000003000F0000000000000000400000001
0000000000100100000000000000071C0107E8033A08
0000000000100110000000000000071C0107E8033A08
0000000000000120000000000002C476C6562204B7279616B6874756E6F762C20676C65622E6
B7279616B6874756E6F764079616E6465782E7275
0000000000000130000000000000DD0A3D093D09E20D0A4D0912031
0000000000000140000000000000AD09FD183D182D18C2031
00000000F041A2A475C
```

Рис. 29. Сериализованные данные библиотеки УГО ФБ

```
--------- До операции десериализации ---------
Name: None
CreateTime: 2024.01.28 04:05:36
UpdateTime: 2024.01.28 04:05:36
Author: None
Description: None
Versions: None
--------- После операции десериализации ---------
Name: Тестовая библиотека
CreateTime: 2024.01.28 04:36:36
UpdateTime: 2024.01.28 04:36:36
Author: Anton Boronnikov, boronnikov-anton@mail.ru
Description: Библиотека для отладки разработанного формата
Versions:
 Id: 1 Categories:
  Id: 1
  Name: Категория 1
 Elements:
  Name: УГО ФБ 1
  CategoryId: 1
  CreateTime: 2024.01.28 04:36:36
  UpdateTime: 2024.01.28 04:36:36
  Author: Anton Boronnikov, boronnikov-anton@mail.ru
  Description: УГО ФБ 1
  Source: Путь 1

 Id: 1 Categories:
  Id: 1
  Name: Категория 1
 Elements:
  Name: УГО ФБ 1
  CategoryId: 1
  CreateTime: 2024.01.28 04:36:36
  UpdateTime: 2024.01.28 04:36:36
  Author: Gleb Kryakhtunov, gleb.kryakhtunov@yandex.ru
  Description: УГО ФБ 1
  Source: Путь 1
```

Рис. 30. Результат операции десериализации

VII. Заключение

В настоящей статье рассмотрены подходы к созданию проприетарного формата представления данных.

Выполнен обзор метрик основных существующих форматов хранения данных – JSON, XML, CSV, HDF5, AVRO, Intel HEX, ORC. На основании данного анализа разработана схема данных, позволяющая описать иерархические объекты любой сложности. Преимуществом данной схемы является то, что она не привязана к конкретному языку программированию или типам данных, поэтому она является универсальным решением, в базисе которого можно разработать проприетарный формат любой сложности.

Конфиденциальность данных достигается тем, что разработчик формата самостоятельно определяет кодировку значений идентификаторов, типов записей и структуру данных для предложенной схемы данных. Также схема данных обеспечивает контроль целостности информации путем добавления контрольной суммы CRC в конец файла, параметры которой определяются разработчиком формата.

Данные записи кодируются в кодировке UTF-8, что позволяет описать данные на различных языка мира. Содержимое файла кодируется в шестнадцатеричной системе счисления, где каждый байт кодируется в виде пары символов, что позволяет сделать содержимое файла непонятным обычному пользователю.

На основе предложенного решения был разработан формат LSGS, предназначенного для хранения данных библиотеки УГО ФБ. Представлен результат сериализации и десериализации объектов.

В дальнейшем планируется разработать еще формат на основе предложенной схемы данных, который позволяет сохранить настройки любого вида ПО, написанного на языке программирования высокого уровня.

Библиография

[1] ASCII-Code.com – URL: https://www.ascii-code.com/ (дата обращения: 19.01.2024)

[2] Unicode – URL: https://home.unicode.org/ (дата обращения: 19.01.2024)

[3] Кодировка символов Windows-1251 – URL: https://wm-school.ru/html/html_win-1251.html (дата обращения: 19.01.2024)

[4] Understanding and implementing CRC (Cyclic Redundancy Check) calculation – URL: https://www.sunshine2k.de/articles/coding/crc/understanding_crc.html (дата обращения: 20.01.2024)

[5] Базикалов И.В., Поплавская В.А. Сравнительный анализ распространённых хэш-функций. // Студенческая наука для развития информационного общества. Сборник материалов VI Всероссийской научно-технической конференции. — 2017. — Т.2. — с. 230-233.

[6] Комарова А.В., Менщиков А.А., Коробейников А. Г. Анализ и сравнение алгоритмов электронной цифровой подписи ГОСТ р 34. 10-1994, ГОСТ р 34. 10-2001 и ГОСТ р 34. 10-2012. // Вопросы кибербезопасности. — 2017. — №1(19). — с. 51-56.

[7] Введение в JSON – URL: https://www.json.org/json-ru.html (дата обращения: 23.01.2024)

[8] Наумов Р.К., Железков Н.Э. Сравнительный анализ форматов хранения текстовых данных для дальнейшей обработки методами машинного обучения. // Научный результат. Информационные технологии. — 2021. — Т.6, №1. — с. 40-47.

[9] Белов В.А., Никульчев Е.В. Экспериментальная оценка временной эффективности обработки больших данных в заданных форматах хранения. // International Journal of Open Information Technologies. — 2021. – Т.9, №9. — с. 95-102.

[10] Что такое XML? – URL: https://aws.amazon.com/ru/what-is/xml/ (дата обращения: 24.01.2024)

[11] Никифоров Д.А., Корж Д.В., Сиваков Р.Л. Обзор инструментов для валидации XML-документов с помощью правил контроля, описанных на объектном языке ограничений (OCL). // Информационные технологии. — 2017. — Т.23, №5. — с. 342-351.

[12] CSV File Format Specification – URL: -https://arquivo.pt/wayback/20160521044400/http://mastpoint.curzonnassau.com/csv-1203/ (дата обращения: 24.01.2024)

[13] HDF5 – URL: https://www.hdfgroup.org/solutions/hdf5/ (дата обращения: 25.01.2024)

[14] Apache Avro™ 1.11.1 Documentation – URL: https://avro.apache.org/docs/1.11.1/ (дата обращения: 25.01.2024)

[15] Формат Intel-HEX – URL: https://spd.net.ru/Article/Intel-HEX (дата обращения: 25.01.2024)

[16] Apache ORC – URL: https://orc.apache.org/docs/ (дата обращения: 26.01.2024)

[17] Unified modeling language – URL: https://www.uml.org/ (дата обращения: 26.01.2024)

Approaches to creating a proprietary data representation format

G.M. Kryakhtunov, A.S. Boronnikov

***Abstract* — While developing software, the question inevitably arises about choosing a data presentation format, that determines how information will be organized for efficient data storage, transmission and processing. However, well-known universal formats are not always suitable for specific requirements and criteria, so there is a need to develop your own.**

This article discusses the issues of creating a data representation scheme for the development of proprietary data representation formats.

The paper analyzes the main metrics of data storage formats, as a result of which criteria for the data schema are proposed. The main advantages of data storage formats are highlighted during a brief overview of existing formats. Based on the information received, a data schema has been developed for a proprietary data representation format, that implements a hierarchical structure with the proposed classes.

Their structure, the general file recording format, recording types and encoding are defined. In addition, the data schema provides information integrity control by adding a CRC checksum to the end of the file.

As an example of the application of the developed data schema, the paper presents the creation of a closed proprietary LSGS (Library of symbolic graphic symbols) format designed to store data from a library of conditional graphical designations of functional blocks. The result of serialization and deserialization of objects in a high-level programming language is presented.

***Keywords* — data storage format, data representation scheme, data schema, format, relationships, data patterns, LSGS, JSON, XML, CSV, HDF5, AVRO, Intel HEX, ORC.**

REFERENCES

[1] ASCII-Code.com [Online]. Available: https://www.ascii-code.com/
[2] Unicode [Online]. Available: https://home.unicode.org/
[3] Windows-1251 character encoding [Online]. Available: https://wm-school.ru/html/html_win-1251.html
[4] Understanding and implementing CRC (Cyclic Redundancy Check) calculation [Online]. Available: https://www.sunshine2k.de/articles/coding/crc/understanding_crc.html
[5] Bazikalov I.V., Poplavskaya V.A. Comparative analysis of common hash functions // Student science for the development of the information society. Collection of materials of the VI All-Russian Scientific and Technical Conference. — 2017. — Vol. 2. — pp. 230-233 (in Russian).
[6] Komarova A.V., Menshchikov A.A., Korobeinikov A.G. Analysis and comparison of electronic digital signature algorithms GOST r 34. 10-1994, GOST r 34. 10-2001 and GOST r 34. 10-2012 // Cybersecurity issues. — 2017. — No. 1(19). — pp. 51-56 (in Russian).
[7] Introduction to JSON [Online]. Available: https://www.json.org/json-ru.html
[8] Naumov R.K., Zhelezkov N.E. Comparative Analysis of Text Data Storage Formats for Further Processing Using Machine Learning Methods // Scientific result. Information Technology. — 2021. — Vol. 6, No. 1. — pp. 40-47 (in Russian).
[9] Belov V.A., Nikulchev E.V. Experimental evaluation of the temporal efficiency of big data processing for specified storage formats // International Journal of Open Information Technologies. — 2021. — Vol. 9, No. 9. — pp. 95-102 (in Russian).
[10] What is XML? [Online]. Available: https://aws.amazon.com/ru/what-is/xml/
[11] Nikiforov D.A., Korzh D.V., Sivakov R.L. An overview of tools for validating XML documents using control rules described in Object Constraint Language (OCL) // Information Technology. — 2017. — Vol. 23, No. 5. — pp. 342-351 (in Russian).
[12] CSV File Format Specification [Online]. Available: https://arquivo.pt/wayback/20160521044400/http://mastpoint.curzonnassau.com/csv-1203/
[13] HDF5 [Online]. Available: https://www.hdfgroup.org/solutions/hdf5/
[14] Apache Avro™ 1.11.1 Documentation – URL: https://avro.apache.org/docs/1.11.1/
[15] Format Intel-HEX [Online]. Available: https://spd.net.ru/Article/Intel-HEX
[16] Apache ORC [Online]. Available: https://orc.apache.org/docs/
[17] Unified modeling language [Online]. Available: https://www.uml.org/