ISSN 2079-5459 (print)
ISSN 2413-4295 (online)

УДК 004.75 doi:10.20998/2413-4295.2016.12.15

ВИБІР ПРОТОКОЛУ СЕРІАЛІЗАЦІЇ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМУНІКАЦІЙНОГО МОДУЛЯ SCADA-СИСТЕМ

Ю. Є. ГРУДЗИНСЬКИЙ*, *, Р. В. МАРКОВ

Кафедра АТЕП, НТУУ «КПІ», Київ, УКРАЇНА
**email: jug@sonettele.com*

АНОТАЦІЯ *В даній статті розглянуто сучасні протоколи серіалізації даних. Зокрема розглянуто протоколи XML, JSON, упакування в бінарний вигляд, Protobuf та представлення даних у вигляді рядків. Проведено порівняння даних способів серіалізації даних для подальшого використання в розробці програмного забезпечення комунікаційного модуля SCADA-систем. Описано основні переваги та недоліки вище вказаних протоколів серіалізації. Зроблено висновки по доцільності використання Protobuf.*
Ключові слова: *протокол, XML , JSON, Protobuf, серіалізація.*

PROTOCOL SELECTION FOR SERIALIZATION SOFTWARE DEVELOPMENT COMMUNICATION MODULE SCADA-SYSTEMS

Y. GRUDZYNSKYY*, *, R. MARKOV

Department ATEP, NTUU «KPI», Kiev, Ukraine
**email:jug@sonettele.com*

ABSTRACT *The main purpose of this article is a selection protocol serialization data to develop software communication module SCADA - systems. For this purpose, considered serialization protocols that have become popular to date. These protocols are XML, JSON, Protobuf, packaging type and a binary representation of data in rows. In the present article is to compare them, prompting the main advantages and disadvantages.*
Particularly simple and very popular way - is to present all data in rows.The advantage of this approach is that the data are those that can be read by a man, and the implementation is simple and requires no special knowledge. The main disadvantage of this approach is that the structures represent a string suitable only for very simple data sets. Also have to write a lot of code to encode and decode, and speed of execution is poor. Another popular method - a record of data in XML. The data in this case are clearer than in the case of simple ASCII strings, and flexibility are on top. But as is the case with the previous method, read and write data in XML-format imposes great limitations on performance.Today's data serialization protocol is protocol as JSON. This is an easy way to store and transfer of structured data, based on the text. With a simple syntax becomes possible to easily store as simple numbers and strings and arrays, objects, using nothing but a text. JSON compact compared to XML, clear and easy for people to read the computer. It can easily be converted into software format.
The disadvantage of the above described protocol is that all facilities provided are for low-level tools. Also of great importance is the size of the original package, integrity and security of data when dealing with protocols listed above can not be guaranteed because the data transmission based only on text transformations. All these shortcomings methods designed to eliminate Protobuf by Google.
Keywords*: protocol, XML, JSON, Protobuf, serialization.*

Введення

Комунікаційні модулі сучасної SCADA-системи виконують багато роботи, пов’язаної з обміном інформації між контролерами промислової мережі та самою системою. Зазвичай, ці дані необхідно зберігати в самій системі, а також треба забезпечити ефективний, надійний та швидкий обмін ними між іншими елементами SCADA-системи . Часто в самій програмі ми працюємо з цими даними зовсім в іншому вигляді і доводиться писати багато коду, щоб зберегти/відновити стан об'єктів діючої системи. Для серіалізації даних обміну використовується декілька протоколів.

Серіалізація - це процес переведення будь-якої структури даних в послідовність бітів. Зворотньою до операції серіалізації є операція десеріалізації (структуризації) - відновлення початкового стану структури даних з бітової послідовності. Ця послідовність може бути як бінарним представленням цих даних, так і текстовим. У більшості випадків серіалізація потрібна для пересилання будь-яких повідомлень по мережі.

Мета роботи

Основна мета - це вибір протоколу серіалізації даних для розробки програмного забезпечення комунікаційного модуля SCADA – систем, що дозволить здійснювати швидкий, надійний та паралельний обмін даними.

© Ю. Є. ГРУДЗИНСЬКИЙ, Р. В. МАРКОВ, 2016

Виклад основного матеріалу

Коли перед програмістом постає завдання серіалізації даних, наприклад для їх подальшої передачі по мережі, у нього є кілька шляхів.

Найпростіший і досить популярний спосіб - це представити всі дані у вигляді рядків. У цьому випадку на виході ми отримаємо потік ASCII-символів, який потім буде передано по мережі або записано у файл.

Перевагою цього підходу є те, що дані залишаються такими, що можуть бути прочитані людиною, а сама реалізація проста і не вимагає спеціальних знань.

Основним недоліком даного підходу є те, що представлення структур у вигляді рядка підійде тільки для дуже простих наборів даних. До того ж доведеться писати досить багато коду для кодування і декодування, а швидкість його виконання буде бажати кращого.

Інший популярний метод - це запис даних в XML. XML-документ являє собою звичайний текстовий файл, в якому за допомогою спеціальних маркерів створюються елементи даних, послідовність і вкладеність яких визначає структуру документа і його зміст. Основною перевагою XML документів є те, що при відносно простому способі створення та обробки, вони дозволяють створювати структуровану інформацію.

Різноманітних бібліотек, що займаються парсингом XML-файлів, можна нарахувати безліч, що спрощує процес написання механізмів серіалізації. Дані в цьому випадку представлені наочніше, ніж у випадку простих ASCII рядків, та й гнучкість тут на висоті. Багато популярних протоколів використовують цю мову розмітки в якості своєї основи, так як вона розширювана і дозволяє не сильно замислюватися про зворотну сумісність при оновленні структури даних. Але, як і у випадку з попереднім способом, читання і запис даних в XML-формат накладає великі обмеження на продуктивність. Навігація по дереву вимагає високої потужності процесора, та й код, все-таки, теж доведеться трохи дописати, щоб все працювало так, як задумано. Ще один мінус, який необхідно врахувати - це розмір одержуваних даних. При досить великих обсягах інформації або поганих мережевих з'єднаннях використовувати XML не надто доцільно.

Сучасним протоколом серіалізації даних на сьогодні також являється протокол JSON. JavaScript Object Notation - простий формат обміну даними, зручний для читання і написання як людиною, так і комп'ютером. Він заснований на підмножині мови програмування JavaScript, визначеної в стандарті ECMA-262 3rd Edition - December 1999. JSON - текстовий формат, повністю незалежний від мови реалізації, але він використовує угоди, знайомі програмістам С-подібних мов, таких як C, C ++, C # , Java, JavaScript, Perl, Python і багатьох інших. За допомогою простого синтаксису стає можливим легко зберігати як прості числа і рядки, так і масиви, об'єкти, використовуючи при цьому не що інше як текст. Так само можна пов'язувати об'єкти і масиви, що дозволяє створювати складні структури даних.

JSON має ряд переваг:

- компактніший ніж XML;
- зрозумілий для людей і легко зчитується комп'ютером;
- його легко можна перетворити в програмні формати: числові значення, рядки, булевий формат, масиви і асоціативні масиви.
- майже всі програмні мови мають функції, що дозволяють зчитувати і створювати json формат даних.

JSON складається із таких основних частин:

- Колекція пар ключ/значення. У різних мовах, ця концепція реалізована як об'єкт, запис, структура, словник, хеш, іменований список або асоціативний масив.
- Упорядкований список значень. У більшості мов це реалізовано як масив, вектор, список або послідовність.

Це універсальні структури даних. Майже всі сучасні мови програмування підтримують їх в будь-якій формі. Логічно припустити, що формат даних, незалежний від мови програмування, повинен бути заснований на цих структурах.

Основним недоліком вище розглянутих протоколів є те, що всі засоби, що надаються для роботи, наприклад з JSON чи XML, є низькорівневими інструментами. Окрім того кінцевий розмір серіалізованих даних хотілося б зменшити, оскільки робота комунікаційного модуля SCADA-системи пов’язана з великою інформаційною завантаженістю мережі. Велике значення також має цілісність та захищеність даних, що при роботі із вище наведеними протоколами не може бути гарантована, оскільки передача даних основана лише на текстових перетвореннях.

Останній метод, який набув широкого поширення - це упаковка в бінарний вигляд. В цьому випадку ми практично повністю втрачаємо читабельність серіалізованих даних, але зате значно виграємо в швидкості їх парсинга і вихідному обсязі. Все це звичайно добре, але в разі байтового подання інформації ми отримуємо багато проблем з сумісністю при зміні структур даних в програмі, а також витрачаємо багато зусиль на підтримку коду упаковки і розпаковування в актуальному стані.

ISSN 2079-5459 (print)
ISSN 2413-4295 (online)

Цей недолік та всі недоліки вищеперерахованих методів покликаний усунути протокол Protobuf від Google . Protocol Buffers - це спеціальний метод кодування структур даних, який дозволяє швидко і без проблем серіалізувати все що завгодно. Protocol Buffers має офіційну підтримку від Google для таких мов, як C ++, Java і Python. Ця підтримка виражається в наявності компілятора для спеціальної мови, що описує структури даних. Структура даних визначається заголовними файлами *.proto. Компілятор генерує з них об'єкти на різних мовах програмування, що забезпечує властивість кросплатформенності.

Такий підхід, з одного боку, дозволяє замінити будь-який компонент системи на його реалізацію на іншій мові, а, з іншого, дозволяє зручно підтримувати різні версії об'єктів, додавати нові поля і змінювати існуючі. Google розробив Protobuf для використання в своїх внутрішніх службах. Це бінарний формат кодування, який дозволяє задати схему для даних, що серіалізуються, з використанням спеціальних специфікацій.

Protobuf дозволяє описувати прості структури даних на спеціальній мові, яка потім компілюється в класи, що представляють ці структури. Разом з класами йде оптимізований код їх серіалізації в компактний формат представлення. А найкраще те, що доступ до даних максимально спрощено, оскільки для доступу до кожного поля у класі є відповідні методи "get" і "set", а для серіалізації об'єкта в масив байтів або потік введення/виведення потрібно зробити всього один виклик функції.

Protocol Buffers дозволяє перевести дані і об'єкти в програмі в бінарний вид і безпечно переслати їх по мережі або зберегти у файл для подальшого використання. Основною перевагою Protocol Buffers є простота використання. Досить зробити метаопис об'єкта, після чого виходить готовий код об'єкта і його серіалізація. Користувачеві потрібно просто використовувати отриманий код. Стандартні засоби Protocol Buffers дають можливість не тільки виконувати плоску (послідовну) серіалізацію, а й серіалізацію в потік, що дозволяє відразу вивести дані в файл, мережу або кудись ще і дозволяє підтримувати нормальні C++ потоки типу std::ostream і std::istream.

Почати роботу з протоколом Protobuf дуже просто. Спершу слід описати дані в proto-файлі. Розглянемо роботу із даним протоколом на прикладі програми що працює з списком людей і номерами їх кредитних карт. Мовою protobuf необхідні структури даних будуть виглядати приблизно так:

```
package CardsApp;

message CardHolder {
   required string firstName = 1;
   required string lastName = 2;
   required int32 id = 3;

   enum CardType {
      VISA = 0;
      MASTERCARD = 1;
      AMERICANEXPRESS = 2;
   }

   message CreditCard {
      required string cardNumber = 1;
      optional CardType type = 2 [default = VISA];
   }

   repeated CreditCard card = 4;
}

message CardHoldersList {
   repeated CardHolder person = 1;
}
```

На перший погляд код дуже схожий на програму написану на C++. І дійсно, синтаксис дуже схожий. На початку файлу знаходиться рядок з ім'ям пакета. Він визначає зону видимості даних і служить для запобігання конфлікту імен. Далі йде блок з повідомленнями, які починаються з ключового слова message. Ці конструкції є аналогами структур в C ++. Поля повідомлення підтримують такі типи даних, як bool, string, int32 і так далі. Наприклад, поле firstName є строковою змінною. На початку оголошення цієї змінної знаходиться ключове слово required. З назви неважко здогадатися, що це поле повинно бути завжди ініціалізоване. Всього таких специфікаторів може бути три: required, optional і repeated. Optional говорить protobuf-компілятору, що поле може бути не ініціалізоване, а repeated повідомляє про можливість неодноразового повторення змінної, описаної за допомогою цього специфікатор в структурі даних. Крім того, кожен елемент має так звані теги.

У прикладі видно, що, крім повідомлень, ми можемо визначати перерахування, а самі messages можуть бути вкладені одна в другу. Все це обіцяє нам високу гнучкість і легкість при роботі з кодом.

Після правильного опису використовуваних нами даних надаємо наш протофайл спеціальному компілятору. Оскільки Google офіційно підтримує цілих три мови програмування, то компілятору треба знати, під яку мову ми генеруємо код. Якщо наш файл називається cardholders.proto, то для успішного завершення операції командний рядок буде виглядати приблизно так:

```
protoc -I=$SRC_DIR --cpp_out=$DST_DIR $SRC_DIR/cardholders.proto
```

Тут варто звернути увагу на параметр -cpp out, саме він визначає, що ми генеруємо код для C++.

Кожен клас Protocol Buffer має функції серіалізації в бінарне представлення і парсинга цього подання.

```
bool SerializeToString(string* output) const;
bool ParseFromString(const string& data);
bool SerializeToOstream(ostream* output) const;
bool ParseFromIstream(istream* input);
```

Тут слід зауважити, що serialize-методи використовують STL-рядок лише в якості контейнера для бінарних даних. Не варто сподіватися, що, заглянувши всередину string-змінної, можна буде виявити хоч який-небудь текст, що читається.

Використання отриманих класів на практиці є дуже простим. Щоб переконатися в цьому, досить поглянути на код нижче.

```
int main(int argc, char* argv[]) {
 GOOGLE_PROTOBUF_VERIFY_VERSION;
 CardsApp::CardHoldersList card_holders;
 AddToCardHolders(card_holders.add_person());

 fstream output(argv[1], ios::out | ios::trunc | ios::binary);
 if (!address_book.SerializeToOstream(&output)) {
   cerr << "Failed to write file." << endl;
   return -1;
  }

 fstream input(argv[1], ios::in | ios::binary);
 if (!input) {
   cout << argv[1] << ": File not found. Creating a new
file." << endl;
   }
 else if (!card_holders.ParseFromIstream(&input)) {
   cerr << "Failed to parse file." << endl;
    return -1;
    }

 ListCardHolders(card_holders);
 google::protobuf::ShutdownProtobufLibrary();
 return 0;
 }
```

Перше, що ми бачимо в main-функції, - це макрос перевірки версії протобафа - GOOGLE_PROTOBUF_VERIFY_VERSION. Далі ми оголошуємо змінну card_holders, яка буде містити всі записи про власників карт. На поточний момент там порожньо, і ми додаємо туди один запис викликом функції AddToCardHolders, код якої розглянемо трохи нижче. Для збереження всього це в файлі ми відкриваємо файловий потік і викликаємо SerializeToOstream.

Щоб переконатися в тому, що все пройшло нормально, прочитаємо серіалізоване повідомлення зі щойно створеного протобафом файлу. Для цього створимо ще один потік з атрибутами читання і викличемо метод ParseFromIstream. Після цього виведемо на екран прочитане за допомогою написаної нами функції ListCardHolders. Ну а в кінці не забуваємо викликати google::protobuf::ShutdownProtobufLibrary () для запобігання витоку пам'яті.

```
void ListCardHolders(const
CardsApp::CardHoldersList& card_holders) {
  for (int i = 0; i < card_holders.person_size(); i++) {
    const CardsApp::CardHolder& person =
       card_holders.person(i);
    cout << "Person ID: " << person.id() << endl;
    cout << "First Name: " << person.firstName() <<
       endl;
    cout << "First Name: " << person.lastName() <<
       endl;
    }
  }

void AddToCardHolders(const CardsApp::CardHoldert&
card_holder) {
  cout << "Enter person ID number: ";
  int id;
  cin >> id;
  card_holder->set_id(id);
  cin.ignore(256, '\n');

  cout << "Enter first name: ";
  getline(cin,*card_holder-mutable_firstName());

  cout << "Enter last name: ";
  getline(cin,*card_holder->mutable_lastName());
  }
```

Ну і наостанок поглянемо на код функцій ListCardHolders і AddToCardHolders. Доступ до елементів повідомлення здійснюється за допомогою GET-методів, імена яких збігаються з іменами самих елементів. Запис значень проводиться за допомогою set_ методів.

Висновки

Із описаних вище протоколів серіалізації даних, можна зробити висновок, що для того щоб забезпечити ефективну роботу комунікаційного модуля SCADA системи (тим самим забезпечивши швидку, паралельну і надійну передачу даних), доцільно і обґрунтовано буде використовувати при розробці програмного забезпечення протокол Protobuf. Цей висновок підтверджено при створенні прототипу комунікаційного модуля нової вітчизняної SCADA-системи.

Список літератури

1. PROTOBUF VS. BOOST::SERIALIZATION [Електронний ресурс] // журнал "ХАКЕР". – 2013. – Режим доступу до ресурсу:

ISSN 2079-5459 (print)
ISSN 2413-4295 (online)

https://xakep.ru/2013/10/31/protobuf-vs-boost-serialization/.
2. **Еллоіт Р.** XML. Справочник / **Р. Еллоіт, С. Мінс**. – М.: Символ-Плюс. – 2001. – 576 с.
3. JSON и XML. Что лучше? [Електронний ресурс]. – 2007. – Режим доступу до ресурсу: https://habrahabr.ru/post/31225/.
4. 5 Reasons to Use Protocol Buffers Instead of JSON For Your Next Service [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: http://blog.codeclimate.com/blog/2014/06/05/choose-protocol-buffers/.
5. Если вы еще используете JSON, то Google protobuf идет к вам! [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: http://knzsoft.blogspot.com/2012/11/protobuf.html.
6. Сопоставление JSON и XML [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://msdn.microsoft.com/ru-ru/library/bb924435(v=vs.110).aspx.
7. Введение в JSON [Електронний ресурс]. – 2010. – Режим доступу до ресурсу: http://json.org/json-ru.html.
8. Как сериализовать и десериализовать данные JSON [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://msdn.microsoft.com/ru-ru/library/bb412179(v=vs.110).aspx.
9. Json или «Туда и Обратно» [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://habrahabr.ru/company/naumen/blog/228279/.
10. Protocol Buffers [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://developers.google.com/protocol-buffers/.
11. ProtoBuf.js vs JSON [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://github.com/dcodeIO/protobuf.js/wiki/ProtoBuf.js-vs-JSON.
12. COMPARING PROTOBUF, JSON, BSON, XML WITH .NET FOR FILE STREAMS [Електронний ресурс] // 2014 – Режим доступу до ресурсу: http://damienbod.com/2014/01/09/comparing-protobuf-json-bson-xml-with-net-for-file-streams/.
13. Protocol Buffer Basics: C++ [Електронний ресурс] // 2016 – Режим доступу до ресурсу: https://developers.google.com/protocol-buffers/docs/cpptutorial#why-use-protocol-buffers.
14. Google Protocol Buffers in action (C++) [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: http://forums.4fips.com/viewtopic.php?f=3&t=807.

Bibliography (transliterated)

1 PROTOBUF VS. BOOST: SERIALIZATION [Web] // "hacker " magazine ["Hacker" magazine], 2013, https://xakep.ru/2013/10/31/protobuf-vs-boost-serialization/.
2 **Elloit R., Means, S.** XML. Spravochnik. [Directory]. Moskow: Symbol - Plus, 2001, 576 p.
3 JSON i XML. Chto luchshe? [JSON and XML. What's better?] [Web], 2007, https://habrahabr.ru/post/31225/.
4 Reasons to Use Protocol Buffers Instead of JSON For Your Next Service [Web], 2014 http://blog.codeclimate.com/blog/2014/06/05/choose-protocol-buffers/.
5 Esli vy eshche ispol'zuete JSON, to Google protobuf idet k vam! [If you are still using JSON, then Google protobuf goes to you!] [Web], 2012, http://knzsoft.blogspot.com/2012/11/protobuf.html.
6 Sopostavlenie JSON i XML [Comparison of JSON and XML] [Web], 2014 https://msdn.microsoft.com/ru-ru/library/bb924435(v=vs.110).aspx.
7 Vvedenie v JSON [Introduction to JSON] [Web], 2010, http://json.org/json-ru.html.
8 Kak serializovat' i deserializovat' dannye JSON [How to serialize and deserialize data JSON] [Web], 2012, https://msdn.microsoft.com/ru-ru/library/bb412179(v=vs.110).aspx
9 Json ili «Tuda i Obratno» [Json or "There and Back Again"] [Web], 2014, https://habrahabr.ru/company/naumen/blog/228279/.
10 Protocol Buffers [Web], 2016, https://developers.google.com/protocol-buffers/.
11 ProtoBuf.js vs JSON [Web], 2015 https://github.com/dcodeIO/protobuf.js/wiki/ProtoBuf.js-vs-JSON.
12 COMPARING PROTOBUF, JSON, BSON, XML WITH .NET FOR FILE STREAMS [Web], 2015, http://damienbod.com/2014/01/09/comparing-protobuf-json-bson-xml-with-net-for-file-streams/
13 Protocol Buffer Basics: C++ [Web], 2016, https://developers.google.com/protocol-buffers/docs/cpptutorial#why-use-protocol-buffers.
14 Google Protocol Buffers in action (C++) [Web], 2012, http://forums.4fips.com/viewtopic.php?f=3&t=807.

Відомості про авторів (About authors)

Грудзинський Юліан Євгенович – старший викладач, Національний технічний університет України «Київський політехнічний інститут», старший викладач кафедри автоматизації теплоенергетичних процесів теплоенергетичного факультету; м. Київ, Україна; e-mail: jug@sonettele.com.

Grudzynskyy Yulian Yevgenovych – Senior Lecturer, National Technical University of Ukraine "Kyiv Polytechnic Institute", a senior lecturer in the automation processes of heat and power cogeneration faculty; Kyiv, Ukraine; mail: jug@sonettele.com.

Марков Роман Валерійович – студент Національного технічного університету України «Київський політехнічний інститут», група ТА-41м, кафедра автоматизації теплоенергетичних процесів теплоенергетичного факультету; м. Київ, Україна; e-mail: markovovchok12@ukr.net.

Markov Roman Valeriyovich – student National Technical University of Ukraine "Kyiv Polytechnic Institute ", group ТА-41м in the automation processes of heat and power cogeneration faculty; Kyiv, Ukraine; mail: markovovchok12@ukr.net.

Будь ласка посилайтесь на цю статтю наступним чином:

Грудзинський, Ю. Є. Вибір протоколу серіалізації для розробки програмного забезпечення комунікаційного модуля scada-систем / **Ю. Є. Грудзинський, Р. В. Марков** // *Вісник НТУ «ХПІ», Серія: Нові рішення в сучасних технологіях*. – Харків: НТУ «ХПІ». – 2016. – № 12 (1184). – С. 106-111. – doi:10.20998/2413-4295.2016.12.15.

Please cite this article as:
Grudzynskyy, Y., Markov, R. Protocol selection for serialization software development communication module scada-systems. *Bulletin of NTU "KhPI". Series: New solutions in modern technologies*. – Kharkiv: NTU "KhPI", 2016, **12** (1184), 106-111, doi:10.20998/2413-4295.2016.12.15.

Пожалуйста ссылайтесь на эту статью следующим образом:
Грудзинский, Ю. Е. Выбор протокола сериализации для разработки программного обеспечения коммуникационного модуля scada-систем / **Ю. Е. Грудзинский, Р. В. Марков** // *Вестник НТУ «ХПИ», Серия: Новые решения в современных технологиях*. – Харьков: НТУ «ХПИ». – 2016. – № 12 (1184). – С. 106-111. – doi:10.20998/2413-4295.2016.12.15.

АННОТАЦИЯ *В данной статье рассмотрены современные протоколы сериализации данных. В частности рассмотрены протоколы XML, JSON , упаковка в бинарный вид , Protobuf и представление данных в виде строк. Проведено сравнение данных способов сериализации данных для дальнейшего использования в разработке программного обеспечения коммуникационного модуля SCADA -систем. Описаны основные преимущества и недостатки вышеуказанных протоколов сериализации . Сделаны выводы по целесообразности использования Protobuf.*
Ключевые слова: *протокол , XML, JSON , Protobuf , сериализация.*

Надійшла (received) 08.03.2016