Создание неустаревающих программ для манипуляции данными ГНСС в формате GREIS

И. М. Алёшин, К. И. Холодков

Институт физики Земли им. О. Ю. Шмидта РАН, Москва, 123242, Россия
E-mails: ima@ifz.ru, keir@ifz.ru

В работе описан набор инструментов для разработки приложений для манипуляции данными измерений ГНСС в формате GREIS. При создании набора инструментов реализованы механизмы, препятствующие его устареванию за счёт автоматизации процессов генерации исходного кода, составления документации, сборки и тестирования компонентов программного обеспечения (ПО). Это позволяет отнести созданное ПО к разряду неустаревающего (evergreen software). В качестве примера использования набора приведено описание инструмента анализа содержимого файлов измерений ГНСС, получаемых приёмниками фирмы Javad GNSS (и совместимыми). Данная программа использует ключевые структуры и методы из представляемого набора, в частности касающиеся загрузки данных и перебора сообщений. Анализ этой простой программы позволит потенциальному пользователю быстро начать разработку собственного прикладного приложения. Набор инструментов разработки является программным продуктом с открытым исходным кодом, который представлен в репозитории https://github.com/iperas/Greis. Инструмент анализа также выпущен под открытой лицензией и доступен по адресу https://github.com/iperas/Greistools.

Ключевые слова: ГНСС, ГНСС-измерения, манипуляция данными, JPS, GREIS, фреймворк, инструменты разработчика, набор инструментов для разработки ПО

Одобрена к печати: 20.12.2018
DOI: 10.21046/2070-7401-2019-16-1-35-45

Введение

В последнее время всё чаще обсуждается феномен старения и обратного ему процесса омоложения программного обеспечения (software aging and rejuvenation) (см., например, (Cotroneo et al., 2011)). Одной из основных причин старения является быстрый рост технологии, в которой используются программы (Abdullah et al., 2015), что имеет прямое отношение к глобальным навигационным спутниковым системам (ГНСС), которые подвержены практически непрерывным изменениям, обусловленным развитием этих систем. Модификации ГНСС могут быть весьма значительными: изменения набора передаваемых сигналов, замена спутников в составе группировок, наращивания числа спутников и ввод в эксплуатацию новых систем. Так, за последние 10 лет в системе NAVSTAR/GPS в дополнение к сигналам L1 и L2 были введены новые сигналы L1C, L2C и L5; система ГЛОНАСС, помимо первоначальных частотно-модулированных сигналов, начала использование трёх новых сигналов с импульсно-кодовой модуляцией; система COMPASS/Beidou стала работать глобально; запущена в опытную эксплуатацию система GALILEO. Такого рода изменения космического сегмента ГНСС приводят к необходимости практически непрерывной модификации приёмного оборудования. Аппаратная и программная модификация приёмников, как следствие, влечёт за собой оперативное внесение изменений в формат хранения и передачи данных (*рис. 1*). Если наземная служба, регистрирующая сигналы ГНСС, ориентирована на получение максимально полных и актуальных данных, то столь кардинальные изменения оборудования не могут не привести к «старению» программных компонент. Для поддержания нормального функционирования системы необходимо внести изменения в программное обеспечение (ПО), которые приведут к его «омоложению». Практически это означает приведение кода программных компонент в соответствие с изменённым форматом принимаемых данных.

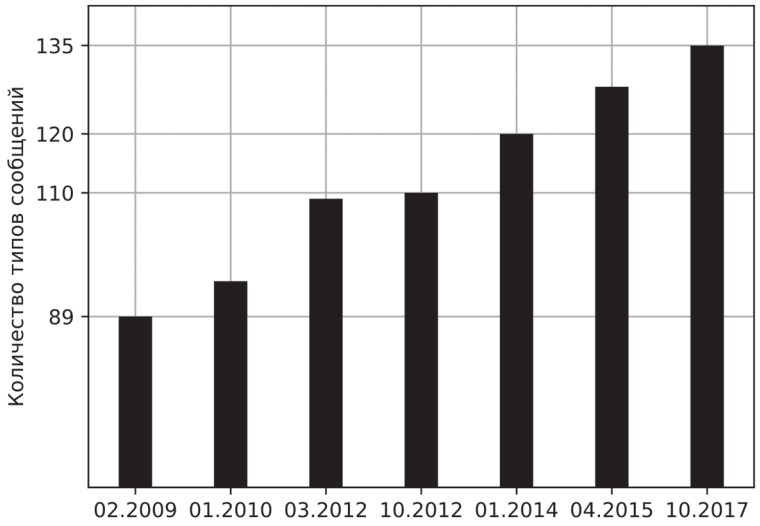

Рис. 1. Динамика роста количества типов сообщений (без учёта вложенных структур) в спецификации GREIS по избранным ревизиям этой спецификации с 2009 по 2017 г.

В ИТ-индустрии ситуации такого рода возникли достаточно давно, и их круг продолжает расширяться. В первую очередь следует упомянуть автоматическое распространение обновлений операционных систем, браузеров, почтовых клиентов, вызванное главным образом оперативной реакцией на обнаружение уязвимостей в системе, устранением ошибок, увеличением быстродействия программ. Выпуски обновлений ПО или его компонентов появляются десятки раз в неделю. Необходимость организации непрерывного оперативного внесения изменений в программный код привело к возникновению концепции «неустаревающего программного обеспечения» (evergreen software) — протологизм, который означает «быть всегда актуальным, соответствующим современным требованиям». В настоящей работе приведено описание разработанного нами неустаревающего набора инструментов, предназначенного для создания ПО, манипулирующего данными ГНСС.

Формат данных, используемый регистрирующей аппаратурой, содержит более полутора сотен структур, которые описывают показания измерений, параметры приёмника и другую информацию (Springer..., 2017). Смена версии встроенного программного обеспечения (firmware) приёмника может затронуть до трети этих структур, что требует внесения соответствующих изменений в код пользовательских программ, обеспечивающих взаимодействие с приёмным оборудованием. Объём таких изменений может быть весьма значительным, что существенно сказывается на времени модификации компонентов. Для увеличения скорости «омоложения» программ максимально широко используются методы и приёмы, позволяющие автоматизировать как можно большую часть процедур, необходимых для модификации, отладки и распространения обновлённых версий этих программ. В настоящее время практически во всех крупных разработках ПО автоматизировано проведение тестов, создание программных интерфейсов и документации. В нашем случае применение этих технологий также существенно упрощает процесс адаптации ПО, но не решает проблему, так как всё ещё требуется ручное изменение исходного кода программ — наиболее трудозатратная часть работы. Конечно, полная автоматизация процесса программирования в имеющемся случае недостижима, однако в нашей задаче частичная автоматизация оказалась возможной благодаря специфике формата данных, используемых в оборудовании Javad GNSS. Из-за особенностей маркетинговой политики крупных производителей приёмного оборудования (GNSS..., 2017) мы вынуждены были ограничиться приёмниками от Javad GNSS, так как только этот производитель предоставляет полную спецификацию используемого формата данных.

Ниже приведено описание созданного нами неустаревающего набора инструментов для разработки ПО (evergreen application framework), ориентированного на создание программ, манипулирующих данными измерений ГНСС, которое является развитием решений, исполь-

зованных нами ранее при создании системы сбора данных сетью наземных станций ГНСС (Aleshin et al., 2014). Набор инструментов разработки предназначен для упрощения создания приложений, связанных с получением и передачей данных ГНСС, особенно в ситуациях, когда инструментов, предоставляемых производителями регистрирующей аппаратуры, недостаточно. В качестве примера можно привести необходимость оперативной обработки и/или публикации данных ГНСС (в частности, при большой частоте опроса) для целей мониторинга значений одного или нескольких регистрируемых параметров в реальном времени. Для решения таких задач в составе набора инструментов имеется программный интерфейс приложений (Application Program Interface, API) для языка C++. В частности, используя этот программный интерфейс достаточно просто решать типичные задачи регистрации данных, такие как чтение сообщений из порта приёмника, манипулирование значениями и последующая запись. Ещё одной важной частью разработки является реализованная нами возможность представления информации в виде таблиц реляционной базы данных. Использование средств из арсенала систем управления базами данных (СУБД) позволяет существенно упростить поиск и отбор сообщений, а также значительно облегчает работу с большими объёмами данных, включая горизонтальное масштабирование, резервное копирование и архивирование.

В первой части статьи описан принцип работы автоматической генерации кода. Во второй части основное внимание уделено описанию интерфейса C++, структур базы данных и программной документации. В третьей части приведён простой пример создания программы с использованием описываемых инструментов.

Автоматическая генерация программной библиотеки

Основу набора инструментов для разработки ПО составляет описание формата представления данных в приёмном оборудовании от Javad GNSS и некоторых приёмниках фирмы TOPCON, ранее производившихся под маркой Javad Positioning Systems (HiPer, Odyssey и др.). Данные в этом формате представлены в форме наборов двоичных последовательностей, каждое из которых мы в дальнейшем будем называть «сообщениями». Идентификация сообщений осуществляется по первым двум байтам, в которых содержится код, однозначно определяющий тип сообщения. Длина сообщения размещена в следующих трёх байтах. Первые пять байтов сообщений мы будем называть заголовком.

Описание всех типов, доступных в текущей версии формата, приведено в спецификации GNSS Receiver External Interface Specification (GREIS) (GREIS..., 2017). Все изменения в программном обеспечении приёмника отображаются в спецификации, т. е. имеется однозначное соответствие версии встроенного программного обеспечения приёмника и версии документа. Это позволяет использовать электронную версию документа в формате PDF (Portable Document Format) для реализации процедуры автоматического омоложения программ. Возможность автоматизации процесса написания программ основана на особенностях синтаксиса, использованного в GREIS, которые позволяют вычленить в документе описание сообщений, формализовать их и преобразовать в машиночитаемый вид. Таким образом, в результате мы получаем полную структурную метаинформацию о формате — набор сущностей, которые содержат формальное описание соответствующих элементов оригинальной спецификации. Форма представления этой метаинформации может быть произвольной. Выбирая из широко распространённых форматов JSON (Java Script Object Notation), YaML (Yet Another Markup Language) и XML (eXtensible Markup Language, XML), мы остановились на последнем, руководствуясь главным образом соображениями удобства, в частности широкой распространённостью библиотек для работы с форматом. Текущая версия встроенного программного обеспечения приёмников поддерживает 134 типа сообщений, каждому из которых соответствует структура в XML-представлении. Полное описание сообщений содержит 154 структуры, так как часть из них имеет вложенные конструкции.

Чтобы обеспечить максимальное быстродействие и возможность создания кроссплатформенных приложений, мы использовали языки программирования C++ и C# — инструмент

платформы .NET (Microsoft Corporation, https://www.microsoft.com/net). Наша задача состоит в создании процедур, которые на основании имеющихся метаданных для каждого типа сообщений обеспечат генерацию соответствующего класса в строгом соответствии с синтаксисом выбранного языка программирования. В результате нам необходимо получить синтаксически верное описание всех данных класса и методов доступа к ним: геттеры и сеттеры (англицизмы от getter и setter), а также конструкторы и деструктор.

Анализ метаданных, полученных в результате разбора документации, показал, что с точки зрения автоматической обработки документа все доступные к моменту написания статьи типы сообщений разбиваются на четыре группы. Каждая из групп требует своих особых методов обработки, но в основе всех соответствующих классов лежит базовый класс, который мы назвали StdMessage, состоящий из трёх членов данных: идентификатор сообщения — целое число, однозначно определяемое его кодом из спецификации; размер тела экземпляра сообщения в байтах; указатель на последовательность байтов, составляющих тело сообщения. Конструктор базового класса получает в качестве аргумента указатели на первый и шестой байты сообщения. Первые пять байтов составляют заголовок — код и размер тела сообщения, которые соответствующим образом интерпретируются конструктором. Это позволяет присвоить значения первым двум членам экземпляра класса, а также выделить память для третьего члена — массива байтов. Работа конструктора завершается побайтовым копированием тела сообщения в выделенный в него участок памяти. Созданный таким образом экземпляр базового класса используется как аргумент конструктора одного из четырёх типов сообщений, к описанию которых мы приступаем.

Для описания типов сообщений в GREIS используется C-подобный синтаксис (*рис. 2*). Описание сообщения начинается его кодом и размером, указанным в фигурных скобках. В первую группу, к которой относятся 99 типов (около двух третей от общего числа), входят сообщения, имеющие фиксированную длину. Так, в примере, приведённом на *рис. 2*, код сообщения, содержащего компоненты скорости, имеет тип “VE”, а размер тела сообщения должен составлять 18 байт.

```
[VE] Cartesian Velocity
  struct Vel {18} {
    f4 x, y, z; // Cartesian velocity vector [m/s]
    f4 vSigma; // Velocity SEP [m/s]
    u1 solType; // Solution type
    u1 cs; // Checksum
  };
```

Рис. 2. Фрагмент спецификации — пример сообщения фиксированной длины. Длину тела сообщения составляет сумма длин всех переменных, и они заранее известны

Очевидно, что создание всех компонент соответствующего класса не вызывает сложности: конструктор класса будет интерпретировать байтовый массив базового класса в соответствии с метаданными для сообщений этого типа и выполнит нужные присваивания значений переменным.

```
[cm], [1m], [2m], [3m], [5m], [lm]: Pseudo-range Corrections
  struct PrCorr {2*nSats+2} {
    ! i2 prc[nSats]; // Correction [Seconds * 1e11]
    u1 mode;    // Mode
    u1 cs;    // Checksum
  };
```

Рис. 3. Описание сообщений «Корректировки псевдодальностей», длина тела которых не определена, так как она зависит от числа спутников, сигналы которых отслеживает приёмник в момент создания им этого сообщения

Ко второй группе (42 типа) относятся сообщения, длина которых заранее не определена, но может быть вычислена при получении экземпляра сообщения, например «Корректировки псевдодальностей» (*рис. 3*, см. с. 38). В данном случае каждый экземпляр сообщения будет содержать массив данных, размер которого определяется только при создании этого экземпляра.

Действительно, коль скоро нам известны структура сообщения и полная длина экземпляра сообщения, размер массива может быть вычислен. Пусть экземпляр сообщения имеет длину L байтов и состоит из заголовка размером H байтов, n величин известного размера по F_k байтов и массива, каждый элемент которого имеет размер M байтов. В этом случае число элементов массива m вычисляется по формуле:

$$m = \frac{L - H - \sum_{k=1}^{n} F_k}{M}.$$

Конструктор сообщений, относящихся ко второму типу, кроме присваивания значений переменным фиксированной длины, копирует элементы массива в соответствующую переменную — контейнер, в нашем случае это STL-вектор.

Третью группу (три типа) составляют сообщения, состоящие из двух частей, одна из которых является обязательной, а другая может отсутствовать (*рис. 4*). Обработка таких сообщений также не составляет труда, так как размер обеих частей фиксирован, что позволяет легко определить наличие необязательной части. В документе GREIS начало и конец описания необязательного раздела определяется по специального вида комментариям: “--- Optional data block ---” и “--- End of optional data block ---” соответственно.

```
[NA] GLONASS Almanac
  struct GLOAlmanac {47 | 52} {
    u1 sv; // Satellite orbit slot number within [1...32] []
    i1 frqNum; // Satellite frequency channel number [-7...24] []
    i2 dna; // Day number within 4-year period starting
    // with the leap year []
    f4 tlam; // Time of the first ascending node passage
    // on day ‘dna’ [s]
    u1 flags; // Satellite flags [bitfield]:
    // 0 - health: 1 - healthy SV, as specified
    // by ‘Cn’, 0 - unhealthy
    // 1 - SVs type: 0 - GLONASS, 1 - GLONASS-M
    // 2...7 - reserved
<...>
    u1 n4; // Number of 4-year period []
// --- Optional data block ---
    u1 reserved; // <reserved>
    f4 gammaN; // Rate of coarse satellite clock correction to
               // GLONASS time scale [s/s]
// --- End of optional data block ---
    u1 cs; // Checksum
  };
```

Рис. 4. Фрагмент описания сообщения третьего типа. Наличие двух чисел после кода говорит о существовании необязательной части, описание которой выделено специальными комментариями-маркерами (выделены полужирным начертанием)

Обработка сообщений третьей и первой групп различается лишь тем, что в третьей группе проводится дополнительный анализ размера сообщения.

```
[sP] Extended Spectrum
  struct Spectrum {n*m*4+7} {
    i2 currFrq; // Current frequency [Hz*10^4]
    i2 finalFrq; // Frequency of the last message [Hz*10^4]
    u1 n; // Number of spectra in this message
    u1 m; // Number of spectrum blocks in this message
    ExtSpecData s[m]; // Extended spectrum data
    u1 cs; // Checksum
  };
```

Рис. 5. Фрагмент описания сообщения, описывающего «Расширенные данные относительной спектральной плотности». В данном сообщении один из членов представляет собой массив, длина элементов которого не фиксирована

Наконец, четвёртая группа (10 типов) включает сообщения, длина которых определяется по специальным правилам, например «Расширенные данные относительной спектральной плотности» (*рис. 5*). Для полной интерпретации сообщений этого вида необходимо определить два неизвестных априори числа, процедуру определения которых унифицировать затруднительно. В текущей реализации программы-генератора из соображений простоты для описания структур из четвёртой группы используется базовый класс StdMessage.

Набор инструментов для разработки приложений

После обработки всех типов сообщений мы можем получить исходный код библиотеки, который представляет собой набор классов, содержащих методы работы с бинарными потоками в формате GREIS. Автоматическая генерация метаданных в формате XML осуществляется программой, разработанной на языке программирования C# под платформу .NET. Для создания библиотек и исполняемых модулей мы использовали среду разработки Qt (The Qt Company Inc., https://qt.io). Компиляция и сборка исходного кода библиотеки, а также процедура тестирования осуществляются набором сценариев системы автоматизации сборки программного обеспечения CMake (Kitware Inc., https://cmake.org).

Полученная таким образом библиотека, которую мы назвали libgreis, составляет ядро набора инструментов. Основное назначение этого набора состоит в возможности использования удобных программных инструментов при создании приложений, которые требуют активного взаимодействия с данными ГНСС. В первую очередь сюда относится ряд проблем, связанных с чтением и записью сообщений в собственном формате Javad, например в задачах регистрации, хранения и публикации данных. Кроме того, набор инструментов может оказаться полезным при создании программ, функционирующих в реальном времени, так как объём рабочих данных в таких приложениях, как правило, невелик и может быть размещён непосредственно в оперативной памяти.

Ситуация меняется, если требуется произвольный доступ (random access) к сохранённым сообщениям, например при необходимости обращения к данным одновременно несколькими приложениями либо для выполнения выборки в большом объёме сообщений. В таких случаях хранение информации в виде файлов становится неэффективным, поэтому инструментарий набора был расширен: нами была реализовано представление сообщений ГНСС в реляционной базе данных и добавлены функции, обеспечивающие взаимодействие с системой управления базами данных (СУБД) MySQL (Oracle Corporation, https://www.mysql.com).

Структура базы данных, используемой для хранения сообщений, достаточно проста и определяется структурой этих сообщений. Каждому типу, указанному в GREIS, соответствует таблица, в столбцах которой хранятся значения величин, входящих в состав сообщения. При записи скалярных величин используются поля совместимого типа, а векторные величины сохраняются в двоичном виде (см. *рис. 2* и *3*). Для индексации сообщений в базе естественно использовать величину, связанную со временем их получения, тем более что

сами по себе сообщения, кроме нескольких специальных, не содержат временной метки. При работе с приёмниками Javad удобно использовать величину, основанную на его внутреннем временном расписании (time grid), которое параметризуется на основе понятия «эпоха» (см. (GNSS..., 2017, с. 21)).

```
CREATE TABLE `epoch` (
id SERIAL,
unixTime BIGINT UNSIGNED NOT NULL,
         PRIMARY KEY (`id`),
         INDEX `idx_unixTime` (`unixTime`)
       );
```

Рис. 6. Определение полей таблицы, определяющей эпохи — внутренние временные интервалы приёмника Javad

GREIS:

```
[pc], [p1], [p2], [p3], [p5], [pl]: Integer Carrier Phases
    struct SCP {4*nSats+1} {
    ! u4 scp[nSats]; // CP, [cycles/1024]
      u1 cs; // Checksum
    };
```

XML:

```
<StandardMessage Name="SCP" Size="-2"
Title="[pc], [p1], [p2], [p3], [p5], [pl]: Integer Carrier Phases"
Validation="Checksum">
  <Variables>
    <Variable GreisType="u4" Name="scp" RequiredValue="">
      <SizeOfDimensions><Dimension>-2</Dimension></SizeOfDimensions>
      <Comment>CP, [cycles/1024]</Comment></Variable>
    <Variable GreisType="u1" Name="cs" RequiredValue="">
      <SizeOfDimensions />
      <Comment>Checksum</Comment></Variable>
  </Variables>
  <Codes><Code>pc</Code><Code>p1</Code><Code>p2</Code>
<Code>p3</Code><Code>p5</Code><Code>pl</Code></Codes>
</StandardMessage>
```

C++:

```
class SCPStdMessage : public StdMessage    {
  public:
    SCPStdMessage(const char* p_message, int p_length);
    std::vector<Types::u4>& Scp() { return _scp; }// CP, [cycles/1024]
    Types::u1& Cs() { return _cs; }// Checksum
<...>
```

SQL:

```
CREATE TABLE `msg_SCP` (
    id SERIAL, idEpoch BIGINT UNSIGNED NOT NULL,
    idMessageCode BIGINT UNSIGNED NOT NULL, bodySize INT NOT NULL,
    `scp` BLOB,
    `cs` SMALLINT UNSIGNED,
```

Рис. 7. Фрагменты метаинформации (метка “XML”), программного кода (метка “C++”) и схемы таблицы СУБД (метка “SQL”), автоматически созданные генератором по исходному файлу спецификации формата производителя (метка “GREIS”)

Начало и конец эпохи задаются сообщениями специального вида [~~] (Receiver Time) и [::] (Epoch Time) соответственно. Первое из них содержит временной штамп, который присваивается всем сообщениям, полученным в течение этой эпохи. Хотя формат допускает использование различных систем представления времени (UTC, GPS, GLONASS и др.), мы предполагаем, что для временной метки применяется GPS. Для описания эпохи в наборе используется класс Epoch, одна из переменных которого представляет собой вектор указателей на StdMessage — коллекцию сообщений, принадлежащих данной эпохе. Кроме того, класс Epoch содержит метод, который обеспечивает доступ к этой коллекции. Для представления последовательности эпох в памяти используется класс DataChunk, одна из переменных которого — контейнер (std::vector) указателей на экземпляры класса Epoch.

Помимо своего прямого назначения, время начала эпохи используется для индексации записей в базе данных. В терминах реляционной базы данных соответствующая структура может быть выражена SQL-командой, приведённой на *рис. 6* (см. с. 41).

В качестве индекса в таблице эпох используется величина, численно равная количеству миллисекунд, прошедших с 00:00 UTC 1 января 1970 г. до начала эпохи. В таблицах базы, предназначенных для хранения сообщений, идентификатор эпохи используется в качестве внешнего ключа (foreign key). Наличие метаинформации, полученной при автоматическом разборе документа GREIS, позволяет автоматизировать создание скриптов, генерирующих структуру базы данных, подобно созданию исходного кода библиотек (*рис. 7*, см. с. 41).

Поддержка взаимодействия создаваемых приложений с СУБД и потоками данных в формате GREIS потребовала введения в набор инструментов разработки дополнительных конструкций (*рис. 8*). Классы MySqlSource и MySqlSink обеспечивают обмен информацией между приложением и базой данных. Для взаимодействия с потоком данных в формате GREIS используется класс GreisMessageStream, производный от IBinaryStream — квази-интерфейса к стандартному (std::istream, std::ostream) бинарному потоку, который представляет двоичный поток данных в формате GREIS в виде набора сообщений. Чтение и запись, кодирование и декодирование сообщений осуществляется соответствующими методами классов DataChunk и Epoch. MySqlSource, MySqlSink и GreisMessageStrem взаимодействуют с DataChunk единообразным способом.

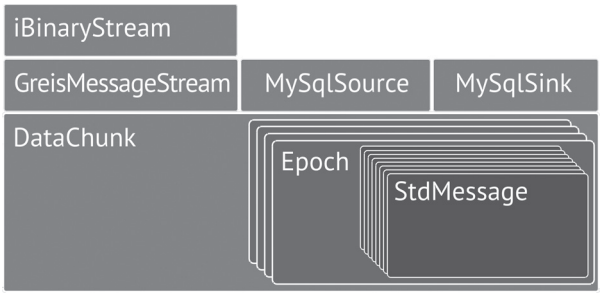

Рис. 8. Обзорная схема основных классов libgreis — библиотеки C++ из набора инструментов. Контейнером верхнего уровня совокупности ГНСС-данных в памяти является класс DataChunk, содержащий вектор указателей на экземпляры класса Epoch. Последние, в свою очередь, содержат контейнеры указателей на сообщения. Для ввода/вывода используются классы, реализующие методы взаимодействия с двоичным представлением (GreisMessageStream) или представлением в реляционной СУБД (MySqlSink, MySqlSource)

Для сборки библиотеки необходимы следующие зависимости: Qt5, libmysql. Для сборки и запуска генератора требуется .Net Framework 4.5.

Применение

Рассмотрим преимущество использования набора инструментов разработки на примере созданной нами программы jpsdump. Назначение программы — вывод временных меток эпох, списка сообщений в порядке появления заголовков и некоторой информации из файла в формате GREIS. Содержание вывода, степень его подробности, а также границы времен-

ного интервала управляются значениями аргументов командной строки. Очевидно, что для выполнения задачи необходимо прочитать и декодировать хранящиеся в файле бинарные сообщения. Как видно из *рис. 9*, использование фреймворка делает это задачу тривиальной.

```
GreisMessageStream
  stream(std::make_shared<FileBinaryStream>(inFileName), true);

auto chunk = make_unique<DataChunk>();
bool hasMore;// Контроль наличия данных в буфере
do {
   hasMore=chunk->ReadBody(stream, MAX_BUF);// Чтение сообщений в буфер
   for (auto& epoch : chunk->Body()) {// Итерация по эпохам
// Вывод временной метки
// эпохи ...
std::cout <<
  QString("Epoch %1:{").\
   arg(epoch->DateTime.toString(Qt::ISODate)).\
    toStdString();
// ... и кодов сообщений
  for (auto& msg : epoch->Messages){// Итерация по сообщениям
// в эпохе
    auto sMsg = static_cast<Greis::StdMessage*>(msg.get());
    std::cout << QString("[%1]").arg(sMsg->Id().c_str()).toStdString();
   <...>
  }
  <...>
   }
} while(hasMore)
```

Рис. 9. Фрагмент программного кода утилиты jpsdump. Полужирным шрифтом выделены переменные и методы, относящиеся к программной библиотеке из набора

Доступ к файлу осуществляется с помощью stream — экземпляра класса GreisMessageStream, а для доступа к эпохам и сообщениям в памяти создаётся chunk — экземпляр класса DataChunk. Chunk наполняется данными из stream при помощи своего метода ReadBody, после чего эпохи с сообщениями становятся доступны для манипуляции. Например, для чтения временных меток и идентификаторов сообщений создаётся два цикла: для эпох, где временные метки доступны как экземпляр QDateTime из переменной DateTime, и для сообщений в эпохах, где для доступа к данным в сообщениях производится явное приведение типа структуры в соответствии с типом сообщения. В нашем случае для чтения идентификатора нет необходимости интерпретировать всё сообщение и для этого используется тип StdMessage.

Заключение

Предложенный нами набор инструментов существенно упрощает производство программного обеспечения, ориентированного на работу с данными ГНСС в формате GREIS, так как обеспечивает непосредственный доступ к регистрируемым сообщениям. При поддержке и развитии таких приложений нет нужды тратить ресурсы на отслеживание изменений в спецификации формата, поскольку этот процесс может быть выполнен автоматически на основе штатной документации производителя. Возможность получения актуальной метаинформации о формате сообщений приёмного оборудования позволяет автоматизировать процесс генерации исходных кодов библиотеки и скриптов создания структуры базы данных.

Последующая компиляция и тестирование программ также выполняются автоматически. Кроме того, следует упомянуть, что вместе с обновлением исходных кодов выполняется соответствующее обновление документации.

Тем не менее полностью автоматизировать процесс «омоложения» программного обеспечения оказалось затруднительно. Дело в том, что исходная документация является продуктом фирмы-производителя приёмного оборудования и внутренняя логика планируемых изменений нам неизвестна. Это может привести к изменениям, которые потребуют внесения дополнений в программу-генератор метаинформации. В качестве недавних примеров таких изменений можно упомянуть использование нового ключевого слова union, а также появление в сообщении [GE] структуры с тремя фиксированными размерами. Ещё одно, крайне досадное, обстоятельство связано с появлением в спецификации очевидных опечаток.

Исходный код программ представлен в репозитории по адресу https://github.com/iperas/Greistools, а код набора инструментов доступен по ссылке https://github.com/iperas/Greis.

Авторы благодарят руководство компании JAVAD GNSS R&D за предоставленное разрешение использовать фрагменты документа GREIS для иллюстраций в данной статье. Работа выполнена в рамках бюджетного финансирования Института физики Земли им. О. Ю. Шмидта РАН.

Литература

1. *Abdullah Z. H.*, *Yahaya J.*, *Deraman A.* Towards anti-ageing model for the evergreen software system // Proc. 5th Intern. Conf. Electrical Engineering and Informatics (ICEEI). Denpasar, Bali, Indonesia. 2015. P. 388–393. DOI: 10.1109/ICEEI.2015.7352532.
2. *Aleshin I.*, *Alpatov V.*, *Vasiliev A.*, *Kholodkov K.*, *Burguchev S.* Data Handling in GNSS Receiver Network and Ionosphere Monitoring Service Solution // 2014 Intern. Conf. Engineering and Telecommunication (EnT). Moscow. 2014. P. 122–125. DOI: 10.1109/EnT.2014.32.
3. *Cotroneo D.*, *Natella R.*, *Pietrantuono R.*, *Russo S.* Software aging and rejuvenation: Where we are and where we are going // 3rd Intern. Workshop on Software Aging and Rejuvenation (WoSAR). Hiroshima, Japan. 2011. P. 1–6. DOI: 10.1109/WoSAR.2011.15.
4. GNSS Market Report. Iss. 5. European GNSS Agency, 2017. 100 p. URL: https://www.gsa.europa.eu/system/files/reports/gnss_mr_2017.pdf.
5. GREIS: GNSS Receiver External Interface Specification. Version 3.7.2. JAVAD GNSS, Inc., 2017. 504 p. URL: https://javad.com/downloads/javadgnss/manuals/GREIS/GREIS_Reference_Guide.pdf.
6. Springer Handbook of Global Navigation Satellite Systems / eds. Teunissen P., Montenbruck O. Cham, Switzerland: Springer Intern. Publishing AG, 2017. 1329 p.

Evergreen framework for GREIS-formatted GNSS data manipulation applications

I. M. Aleshin, K. I. Kholodkov

Schmidt Institute of Physics of the Earth RAS, Moscow 123242, Russia
Emails: ima@ifz.ru, keir@ifz.ru

The paper introduces an application framework that enables easy development of applications that process data generated by GREIS-compatible GNSS receivers. The framework utilizes anti-ageing techniques such as automatic code generation, documentation renewal, code compiling and component testing. We think that the way the framework is designed is what makes this piece of software effectively an *evergreen software*. This paper also includes an example of use case of the framework: a simple data inspection software for measurement data produced by GNSS receivers that deliver

GREIS-compatible output. Data inspection tool utilizes key features of the framework such as data loading and message traversing which give potential users sufficient insight to jump-start their own application. The framework's code is released under LGPL. Source code is available at https://github.com/iperas/Greis. Data inspection tool is also LGPL and available at https://github.com/iperas/Greistools.

Keywords: GNSS, GREIS, JPS, data manipulation, evergreen software, GREIS framework

Accepted: 20.12.2018
DOI: 10.21046/2070-7401-2019-16-1-35-45

References

1. Abdullah Z. H., Yahaya J., Deraman A., *Proc. 5th Intern. Conf. Electrical Engineering and Informatics* (*ICEEI*), Denpasar, Bali, Indonesia, 2015, pp. 388–393, DOI: 10.1109/ICEEI.2015.7352532.
2. Aleshin I., Alpatov V., Vasiliev A., Kholodkov K., Burguchev S., Data Handling in GNSS Receiver Network and Ionosphere Monitoring Service Solution, *2014 Intern. Conf. Engineering and Telecommunication* (*EnT*), Moscow, 2014, pp. 122–125, DOI: 10.1109/EnT.2014.32.
3. Cotroneo D., Natella R., Pietrantuono R., Russo S., Software Aging and Rejuvenation: Where We Are and Where We Are Going, *3rd Intern. Workshop on Software Aging and Rejuvenation* (*WoSAR*), Hiroshima, Japan, 2011, pp. 1–6, DOI: 10.1109/WoSAR.2011.15.
4. *GNSS Market Report*, *Issue 5*, European GNSS Agency, 2017, 100 p., URL: https://www.gsa.europa.eu/system/files/reports/gnss_mr_2017.pdf.
5. *GREIS: GNSS Receiver External Interface Specification*, *Version 3.7.2*, JAVAD GNSS, Inc., 2017, 504 p., URL: https://javad.com/downloads/javadgnss/manuals/GREIS/GREIS_Reference_Guide.pdf.
6. *Springer Handbook of Global Navigation Satellite Systems*, Teunissen P., Montenbruck O. (eds.), Cham, Switzerland: Springer Intern. Publishing AG, 2017, 1329 p.