Обновлено 23 ноября 2017

Кравченко Виктор

GSM-модуль SIM800L: часть 3 — USSD, PDU-формат, отправка SMS в PDU-формате (на кириллице)

Цифровые устройства | Arduino | Arduino Lang | Умный дом | Датчики, модули | Протоколы | Микроконтроллеры и мини ПК

Умный дом, модем, arduino, датчик, module, GSM, SIM800, SIM800L, AT, модуль, SMS, USSD, DTMF, PDU

Содержание:

- GSM-модуль SIM800L: самый полный мануал (на примерах с Arduino Uno) - часть 1. Введение
- GSM-модуль SIM800L: часть 2 - программирование, звонки, прием/отправка SMS, управление по SMS
- → GSM-модуль SIM800L: часть 3 - USSD, PDU-формат, отправка SMS в PDU-формате (на кириллице)
- GSM-модуль SIM800L: часть 4 - DTMF, управление при помощи DTMF
- GSM-модуль SIM800L: часть 5 - прием и передача данных по GPRS в процессе написания

Проверено — автор рекомендует:
GSM/GPRS-модуль SIM800L

Как покупать на Aliexpress со скидкой от 5,5%

Видео-инструкция о покупке со скидками на Aliexpress

В этой статье цикла будут подробно описаны вопросы работы с USSD-запросами и получения ответов на них и их обработки, будет описана работа с кодировкой UCS2, в том числе и в USSD-ответах, а также подробно описан процесс отправки SMS-сообщений в PDU-формате на языках отличных от латиницы (кириллица — русский и пр.).

Для полноценной работы с GSM/GPRS-модулем SIM800L понадобится официальный справочник по AT-командам — SIM800 Series AT Command Manual V1.10.pdf (4,01 MB).

У данной статьи есть видеоверсия! Подписывайтесь на канал, чтобы быть в курсе обновлений!

Здесь и далее в статье, в примерах с модулем SIM800L используется одна схема:

USSD-запросы

Часто бывает полезно, чтобы GSM-модуль сам отслеживал состояние баланса SIM-карты и вовремя информировал владельца о приближении окончания денег. Как известно самый простой способ узнать об остатке — отправить USSD-запрос.

*W **USSD** (англ. **U**nstructured **S**upplementary **S**ervice **D**ata) — стандартный сервис в сетях GSM, позволяющий организовать интерактивное взаимодействие между абонентом сети и сервисным приложением в режиме передачи коротких сообщений.*

На заметку: *Если в коде не предусмотрена обработка кодировки UCS2 и для работы используется текстовый режим (Text Mode), то необходимо использовать USSD-команды, возвращающие сообщения не на кириллице.*
Например, стандартный USSD-запрос проверки баланса для оператора Билайн — `*102#` *вернет ответ в кодировке UCS2 вида:*

```
+CUSD: 0, "003100390038002E............02A0033003100390023", 72
```

Для возврата ответа в текстовом режиме должен использоваться другой USSD-запрос — `#102#`*. Он вернет уже понятный ответ:*

```
+CUSD: 0, " Vash balans 198.02 r.
Dlya Vas - nedelya besplatnogo SMS-obsh'eniya s druz'yami! Podkl.: *319#", 15
```

Для отправки USSD-запроса существует команда **AT+CUSD=<n>[,<str>[,<dcs>]]** (по-умолчанию установлена кодировка IRA).

Описание	Команда	Параметр(ы)	Ответ	Пример(ы)
Отправить USSD-запрос	**AT+CUSD=<n>[,<str>[,<dcs>]]** Незапрашиваемое уведомление: **+CUSD: <n>[,<str_urc>[,<dcs>]]**	**<n>** задает статус ответа: **0** — не получать ответ **1** — получать ответ **2** — отменить сеанс **<str>** — строка запроса в кавычках **<dcs>** — схема кодирования данных (целое число, по умолчанию — 0) **<str_urc>** — текст ответ на USSD-запрос	OK	**AT+CUSD=1,"*100#"**

Исполнение команды, в случае корректного её исполнения, вернет ответ **OK**. Но непосредственно USSD-ответ будет получен в виде незапрашиваемого уведомления **+CUSD**. Именно его нужно отслеживать и обрабатывать, когда оно придет. Пример кода:

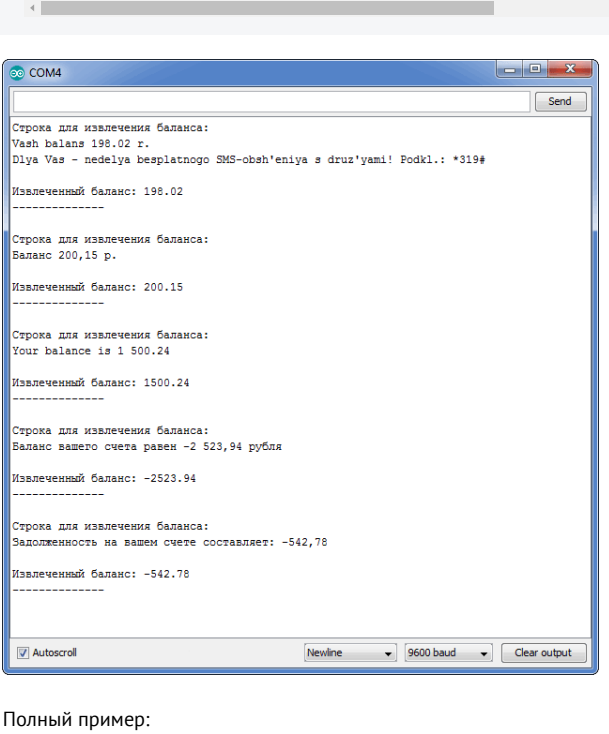
Arduino (C++)

```
#include <SoftwareSerial.h>                   // Библиотека програмной реализации
SoftwareSerial SIM800(8, 9);                  // RX, TX

String _response = "";                        // Переменная для хранения ответа мо
void setup() {
  Serial.begin(9600);                         // Скорость обмена данными с компьют
  SIM800.begin(9600);                         // Скорость обмена данными с модемом
  Serial.println("Start!");

  sendATCommand("AT", true);                  // Отправили AT для настройки скорос

  _response = sendATCommand("AT+CUSD=1,\"*100#\"", true); // Здесь необходимо указат
}

String sendATCommand(String cmd, bool waiting) {
  String _resp = "";                          // Переменная для хранения результат
  Serial.println(cmd);                        // Дублируем команду в монитор порта
  SIM800.println(cmd);                        // Отправляем команду модулю
  if (waiting) {                              // Если необходимо дождаться ответа.
    _resp = waitResponse();                   // ... ждем, когда будет передан отв
    // Если Echo Mode выключен (ATE0), то эти 3 строки можно закомментировать
    if (_resp.startsWith(cmd)) {              // Убираем из ответа дублирующуюся к
      _resp = _resp.substring(_resp.indexOf("\r", cmd.length()) + 2);
    }
    Serial.println(_resp);                    // Дублируем ответ в монитор порта
  }
  return _resp;                               // Возвращаем результат. Пусто, если
}

String waitResponse() {                       // Функция ожидания ответа и возврат
  String _resp = "";                          // Переменная для хранения результат
  long _timeout = millis() + 10000;           // Переменная для отслеживания тайма
  while (!SIM800.available() && millis() < _timeout)  {}; // Ждем ответа 10 секунд,
  if (SIM800.available()) {                   // Если есть, что считывать...
    _resp = SIM800.readString();              // ... считываем и запоминаем
  }
  else {                                      // Если пришел таймаут, то...
    Serial.println("Timeout...");             // ... оповещаем об этом и...
  }
  return _resp;                               // ... возвращаем результат. Пусто,
}

void loop() {
  if (SIM800.available())   {                 // Если модем, что-то отправил...
    _response = waitResponse();               // Получаем ответ от модема для анал
    _response.trim();                         // Убираем лишние пробелы в начале и
    Serial.println(_response);                // Если нужно выводим в монитор порт
    //....
    if (_response.startsWith("+CUSD:")) {     // Пришло уведомление о USSD-ответе
      if (_response.indexOf("\"") > -1) {     // Если ответ содержит кавычки, знач
        String msgBalance = _response.substring(_response.indexOf("\"") + 2);  // По
        msgBalance = msgBalance.substring(0, msgBalance.indexOf("\""));
```

Далее, не составит труда получить необходимую информацию и задать требуемую логику приложения. Функция по «извлечению» состояния баланса из сообщения:

Arduino (C++)

```
void setup() {
  Serial.begin(9600);               // Скорость обмена данными с компьютером
  String balance[5] =  {            // Несколько строк для примера
    "Vash balans 198.02 r.\r\nDlya Vas - nedelya besplatnogo SMS-obsh'eniya s druz'yam
    "Баланс 200,15 р.",
    "Your balance is 1 500.24",
    "Баланс вашего счета равен -2 523,94 рубля",
    "Задолженность на вашем счете составляет: -542,78"
  };
  for (int i = 0; i < sizeof(balance) / sizeof(String); i++) {
    Serial.println("Строка для извлечения баланса:\r\n" + balance[i] + "\r\n");
    Serial.println("Извлеченный баланс: " + (String)getFloatFromString(balance[i]));
    Serial.println("-------------\r\n");
  }
}

float getFloatFromString(String str) {            // Функция извлечения цифр из сообще
  bool   flag     = false;
  String result   = "";
  str.replace(",", ".");                          // Если в качестве разделителя десят

  for (int i = 0; i < str.length(); i++) {
    if (isDigit(str[i]) || (str[i] == (char)46 && flag)) {         // Если начинается г
      if (result == "" && i > 0 && (String)str[i - 1] == "-") {   // Нельзя забывать,
        result += "-";                            // Добавляем знак в начале
      }
      result += str[i];                           // начинаем собирать их вместе
      if (!flag) flag = true;                     // Выставляем флаг, который указывае
    }
    else  {                                       // Если цифры закончились и флаг гов
      if (str[i] != (char)32) {                   // Если порядок числа отделен пробел
        if (flag) break;                          // ...считаем, что все.
      }
    }
  }

  return result.toFloat();                        // Возвращаем полученное число.
}

void loop() {
}
```

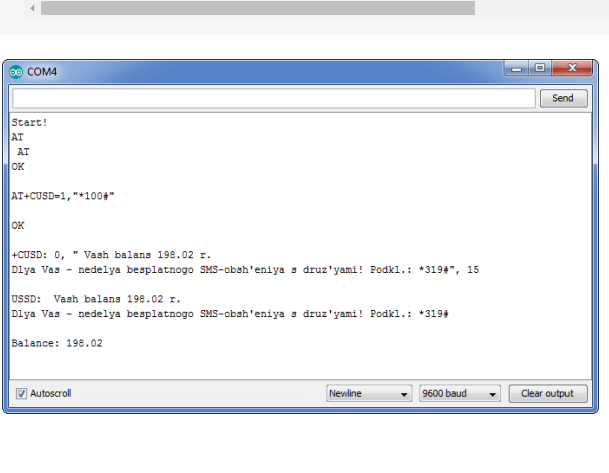


```
Строка для извлечения баланса:
Vash balans 198.02 r.
Dlya Vas - nedelya besplatnogo SMS-obsh'eniya s druz'yami! Podkl.: *319#

Извлеченный баланс: 198.02
-------------

Строка для извлечения баланса:
Баланс 200,15 р.

Извлеченный баланс: 200.15
-------------

Строка для извлечения баланса:
Your balance is 1 500.24

Извлеченный баланс: 1500.24
-------------

Строка для извлечения баланса:
Баланс вашего счета равен -2 523,94 рубля

Извлеченный баланс: -2523.94
-------------

Строка для извлечения баланса:
Задолженность на вашем счете составляет: -542,78

Извлеченный баланс: -542.78
-------------
```

Полный пример:

Arduino (C++)

```
#include <SoftwareSerial.h>                   // Библиотека програмной реализации
SoftwareSerial SIM800(8, 9);                  // RX, TX

String _response = "";                        // Переменная для хранения ответа мо
void setup() {
  Serial.begin(9600);                         // Скорость обмена данными с компьют
  SIM800.begin(9600);                         // Скорость обмена данными с модемом
  Serial.println("Start!");

  sendATCommand("AT", true);                  // Отправили AT для настройки скорос

  // Команды настройки модема при каждом запуске
  //_response = sendATCommand("AT+CLIP=1", true);  // Включаем АОН
  //_response = sendATCommand("AT+DDET=1", true);  // Включаем DTMF
  //_response = sendATCommand("AT+CMGF=1", true);  // Включаем текстовый режим SMS (
  _response = sendATCommand("AT+CUSD=1,\"*100#\"", true); // Здесь необходимо указат
}

String sendATCommand(String cmd, bool waiting) {
  String _resp = "";                          // Переменная для хранения результат
  Serial.println(cmd);                        // Дублируем команду в монитор порта
  SIM800.println(cmd);                        // Отправляем команду модулю
  if (waiting) {                              // Если необходимо дождаться ответа.
    _resp = waitResponse();                   // ... ждем, когда будет передан отв
    // Если Echo Mode выключен (ATE0), то эти 3 строки можно закомментировать
    if (_resp.startsWith(cmd)) {              // Убираем из ответа дублирующуюся к
      _resp = _resp.substring(_resp.indexOf("\r", cmd.length()) + 2);
    }
    Serial.println(_resp);                    // Дублируем ответ в монитор порта
  }
  return _resp;                               // Возвращаем результат. Пусто, если
}

String waitResponse() {                       // Функция ожидания ответа и возврат
  String _resp = "";                          // Переменная для хранения результат
  long _timeout = millis() + 10000;           // Переменная для отслеживания тайма
  while (!SIM800.available() && millis() < _timeout)  {}; // Ждем ответа 10 секунд,
  if (SIM800.available()) {                   // Если есть, что считывать...
    _resp = SIM800.readString();              // ... считываем и запоминаем
  }
  else {                                      // Если пришел таймаут, то...
    Serial.println("Timeout...");             // ... оповещаем об этом и...
  }
  return _resp;                               // ... возвращаем результат. Пусто,
}

void loop() {
  if (SIM800.available())   {                 // Если модем, что-то отправил...
    _response = waitResponse();               // Получаем ответ от модема для анал
    _response.trim();                         // Убираем лишние пробелы в начале и
    Serial.println(_response);                // Если нужно выводим в монитор порт
    //....
```



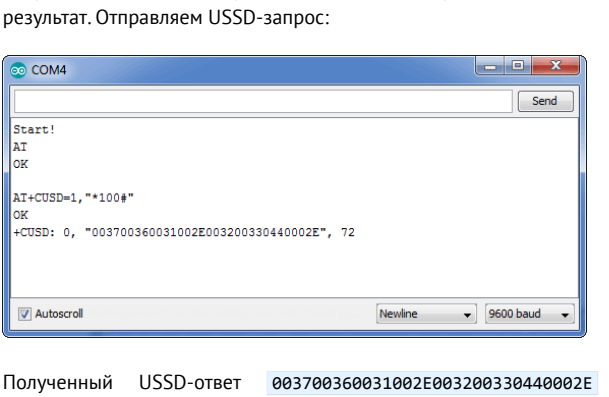
```
Start!
AT
 AT
OK

AT+CUSD=1,"*100#"

OK

+CUSD: 0, " Vash balans 198.02 r.
Dlya Vas - nedelya besplatnogo SMS-obsh'eniya s druz'yami! Podkl.: *319#", 15

USSD:  Vash balans 198.02 r.
Dlya Vas - nedelya besplatnogo SMS-obsh'eniya s druz'yami! Podkl.: *319#

Balance: 198.02
```

Декодирование PDU

*PDU (англ. **P**rotocol **D**escription **U**nit (не Protocol Data Unit)) — один из протоколов передачи SMS-сообщений в GSM-сетях*

Ну, а что же делать абонентам Мегафон, для которых провайдер исключил получение данных в некириллическом формате. Ничего страшного в этом нет.

Рассмотрим на примере отправленного USSD-запроса баланса и полученного ответа, что нужно делать, чтобы получить вменяемый результат. Отправляем USSD-запрос:



```
Start!
AT
OK

AT+CUSD=1,"*100#"
OK
+CUSD: 0, "003700360031002E003200330440002E", 72
```

Полученный USSD-ответ `003700360031002E003200330440002E` представляет из себя строку в кодировке **UCS2** (по сути это первый, устаревший вариант кодировки Unicode спецификации до версии 1.1, не поддерживающий суррогатные символы). В данной кодировке каждый символ имеет фиксированную ширину — 2 байта (16 бит), при этом каждый из байт представлен в HEX-формате. Таким образом каждые четыре знака UCS2-последовательности кодируют всего один символ:

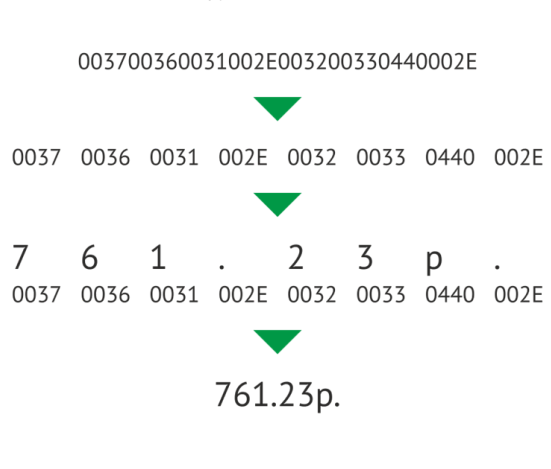
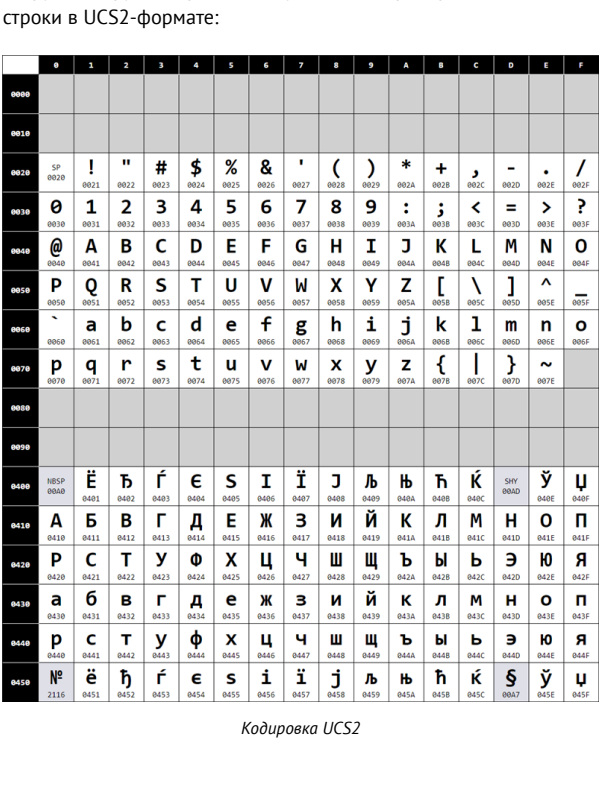

003700360031002E003200330440002E
0037 0036 0031 002E 0032 0033 0440 002E
7 6 1 . 2 3 р .
0037 0036 0031 002E 0032 0033 0440 002E
761.23р.

Таблица кодов UCS2 для кириллических символов. С её помощью нетрудно в ручном режиме осуществить преобразование любой строки в UCS2-формате:



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000																
0010																
0020	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
0080																
0090																
0400	NBSP	Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ	SHY	Ў	Џ
0410	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
0420	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
0430	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
0440	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
0450	№	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	§	ў	џ

Кодировка UCS2

Для того, чтобы автоматически преобразовывать UCS2-строку в читаемый вид, автором написана функция `UCS2ToString()`:

Arduino (C++)

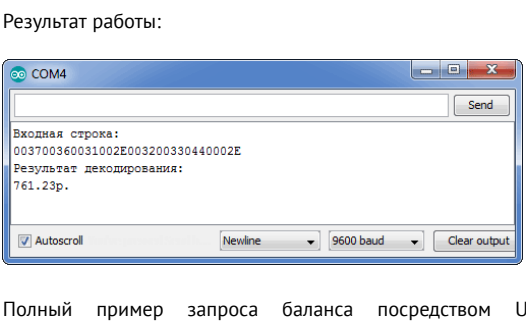
```
String UCS2ToString(String s) {                       // Функция декодирования UCS2 ст
  String result = "";
  unsigned char c[5] = "";                            // Массив для хранения результат
  for (int i = 0; i < s.length() - 3; i += 4) {       // Перебираем по 4 символа кодир
    unsigned long code = (((unsigned int)HexSymbolToChar(s[i])) << 12) +    // Получае
                         (((unsigned int)HexSymbolToChar(s[i + 1])) << 8) +
                         (((unsigned int)HexSymbolToChar(s[i + 2])) << 4) +
                         ((unsigned int)HexSymbolToChar(s[i + 3]));
    if (code <= 0x7F) {                               // Теперь в соответствии с колич
      c[0] = (char)code;
      c[1] = 0;                                       // Не забываем про завершающий н
    } else if (code <= 0x7FF) {
      c[0] = (char)(0xC0 | (code >> 6));
      c[1] = (char)(0x80 | (code & 0x3F));
      c[2] = 0;
    } else if (code <= 0xFFFF) {
      c[0] = (char)(0xE0 | (code >> 12));
      c[1] = (char)(0x80 | ((code >> 6) & 0x3F));
      c[2] = (char)(0x80 | (code & 0x3F));
      c[3] = 0;
    } else if (code <= 0x1FFFFF) {
      c[0] = (char)(0xE0 | (code >> 18));
      c[1] = (char)(0xE0 | ((code >> 12) & 0x3F));
      c[2] = (char)(0x80 | ((code >> 6) & 0x3F));
      c[3] = (char)(0x80 | (code & 0x3F));
      c[4] = 0;
    }
    result += String((char*)c);                       // Добавляем полученный символ к
  }
  return (result);
}

unsigned char HexSymbolToChar(char c) {
  if      ((c >= 0x30) && (c <= 0x39)) return (c - 0x30);
  else if ((c >= 'A') && (c <= 'F'))   return (c - 'A' + 10);
  else                                 return (0);
}
```

Использование функции:

Arduino (C++)

```
void setup() {
  Serial.begin(9600);               // Скорость обмена данными с компьютером

  String UCS2ToDecode = "003700360031002E003200330440002E";
  Serial.println("Входная строка:");
  Serial.println(UCS2ToDecode);
  Serial.println("Результат декодирования:");
  Serial.println(UCS2ToString(UCS2ToDecode));
}

void loop() {
}
```

Результат работы:

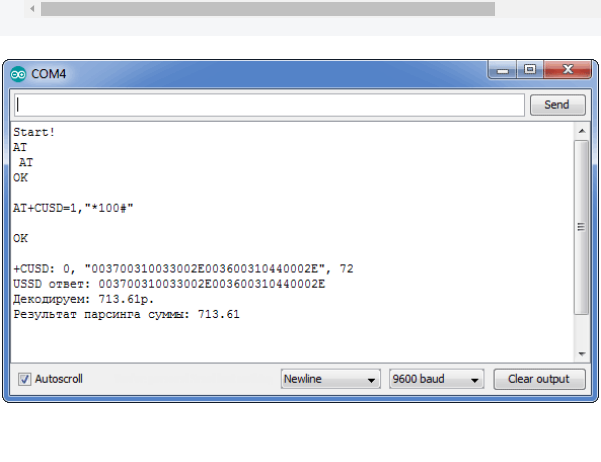


```
Входная строка:
003700360031002E003200330440002E
Результат декодирования:
761.23р.
```

Полный пример запроса баланса посредством USSD, с последующим декодированием ответа и парсингом суммы из полученной строки:

Arduino (C++)

```
#include <SoftwareSerial.h>                   // Библиотека програмной реализации
SoftwareSerial SIM800(8, 9);                  // RX, TX

String _response = "";                        // Переменная для хранения ответа мо
void setup() {
  Serial.begin(9600);                         // Скорость обмена данными с компьют
  SIM800.begin(9600);                         // Скорость обмена данными с модемом
  Serial.println("Start!");

  sendATCommand("AT", true);                  // Отправили AT для настройки скорос
  _response = sendATCommand("AT+CUSD=1,\"*100#\"", true); // Здесь необходимо указат
}

String sendATCommand(String cmd, bool waiting) {
  String _resp = "";                          // Переменная для хранения результат
  Serial.println(cmd);                        // Дублируем команду в монитор порта
  SIM800.println(cmd);                        // Отправляем команду модулю
  if (waiting) {                              // Если необходимо дождаться ответа.
    _resp = waitResponse();                   // ... ждем, когда будет передан отв
    // Если Echo Mode выключен (ATE0), то эти 3 строки можно закомментировать
    if (_resp.startsWith(cmd)) {              // Убираем из ответа дублирующуюся к
      _resp = _resp.substring(_resp.indexOf("\r", cmd.length()) + 2);
    }
    Serial.println(_resp);                    // Дублируем ответ в монитор порта
  }
  return _resp;                               // Возвращаем результат. Пусто, если
}

String waitResponse() {                       // Функция ожидания ответа и возврат
  String _resp = "";                          // Переменная для хранения результат
  long _timeout = millis() + 10000;           // Переменная для отслеживания тайма
  while (!SIM800.available() && millis() < _timeout)  {}; // Ждем ответа 10 секунд,
  if (SIM800.available()) {                   // Если есть, что считывать...
    _resp = SIM800.readString();              // ... считываем и запоминаем
  }
  else {                                      // Если пришел таймаут, то...
    Serial.println("Timeout...");             // ... оповещаем об этом и...
  }
  return _resp;                               // ... возвращаем результат. Пусто,
}

void loop() {
  if (SIM800.available())   {                 // Если модем, что-то отправил...
    _response = waitResponse();               // Получаем ответ от модема для анал
    _response.trim();                         // Убираем лишние пробелы в начале и
    Serial.println(_response);                // Если нужно выводим в монитор порт
    //....
    if (_response.startsWith("+CUSD:")) {     // Пришло уведомление о USSD-ответе
      String msgBalance = _response.substring(_response.indexOf("\"") + 1);  // Полу
      msgBalance = msgBalance.substring(0, msgBalance.indexOf("\""));
      Serial.println("USSD ответ: " + msgBalance);    // Выводим полученный ответ
      // Ответ в UCS2-формате - декодируем и извлекаем число
```



```
Start!
AT
 AT
OK

AT+CUSD=1,"*100#"

OK

+CUSD: 0, "003700310033002E003600310440002E", 72
USSD ответ: 003700310033002E003600310440002E
Декодируем: 713.61р.
Результат парсинга суммы: 713.61
```

Отправка SMS на русском языке (кириллице, и не только) — PDU-формат

*PDU (англ. **P**rotocol **D**escription **U**nit (не Protocol Data Unit)) — один из протоколов передачи SMS-сообщений в GSM-сетях.*

Протокол PDU также очень подробно описан на W Wikipedia.

Раз тема UCS2 уже затронута, нельзя обойти обратную операцию — конвертация обычного текста в UCS2-строку. Такую задачу нужно решать при отправке SMS на языках, отличных от латиницы, на кириллице в том числе. Делается это при помощи представления сообщения в, специально созданном для таких целей, PDU-формате.

Но перед тем, как приступить к самой конвертации, нужно разобрать процедуру отправки SMS в PDU-формате, так как она совершенно отличается от отправки SMS в текстовом формате (Text Mode).

Описание	Команда	Параметр(ы)	Ответ	Пример(ы)
Выбор формата SMS	**AT+CMGF=<mode>**	**<mode>** — формат сообщений, значения: **0** — PDU-формат (по умолчанию) **1** — текстовый формат	OK	**AT+CMGF=0** формат SMS в PDU-формат (PDU Mode)
Отправить SMS в формате PDU	**AT+CMGS=<length><CR><PDU-pack><ctrl-Z/ESC>**	**<length>** — размер сообщения в PDU-пакете **<CR>** — начало строки, после отправки, модуль переходит в режим приема PDU-пакета **<PDU-pack>** — PDU-пакет сообщения После того как текст сообщения передан, необходимо отправить либо **<ctrl-Z>** для отправки сообщения, либо **<ESC>** для отмены.	+CMGS: <n> OK	**AT+CMGS=23** > 0001000B919782198144F400080A04220435044104420021 > +CMGS: 122 OK

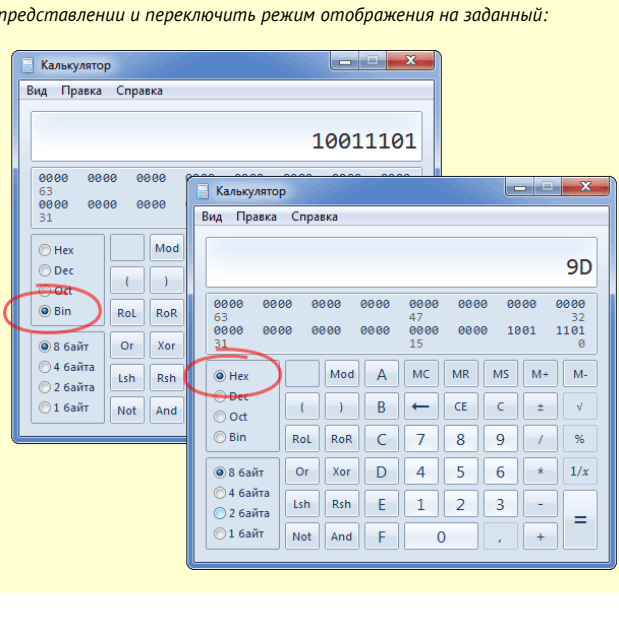
На заметку: *В данном разделе будут использоваться следующие обозначения для представления чисел:*

- `b` *– двоичное представление,*
- `d` *– десятичное представление,*
- `h` *– шестнадцатиричное представление.*

Поскольку PDU-пакет будет формироваться из байт в шестнадцатеричном представлении, каждый из них должен быть представлен двумя символами. Например, шестнадцатиричное представление байта `00001011b` *—* `Bh` *(число 11 в десятичном формате —* `11d`*) должно быть дополнено нулем до двух знаков —* `0Bh`*.*

00001011b = 11d = 0Bh.

На заметку: *Для простой конвертации значений в бинарный (двоичный) / десятичный / шестнадцатиричный формат, можно использовать штатный калькулятор Windows в варианте «Программист». Для этого, необходимо ввести значение в существующем представлении и переключить режим отображения на заданный:*



Для формирования PDU-пакета, необходимо ознакомиться с его структурой — из каких полей он состоит, и какой длины эти поля могут быть. Структура PDU-пакета:

SCA	PDU-type	MR	DA	PID	DCS	VP	UDL	UD
1...12	1	1	2...12	1	1	0, 1 или 7	1	1...140

Описание полей PDU-пакета:

Название поля	Длина, байт	Описание
SCA (**S**ervice **C**enter **A**ddress)	1...12	Номер телефона Центра SMS
PDU Type	1	Тип PDU
MR (**M**essage **R**eference)	1	Порядковый номер сообщения
DA (**D**estination **A**ddress)	2...12	Номер телефона получателя сообщения
PID (**P**rotocol **Id**entifier)	1	Идентификатор протокола
DCS (**D**ata **C**oding **S**cheme)	1	Кодировка сообщения
VP (**V**alidity **P**eriod)	0,1 или 7	Время жизни SMS
UDL (**U**ser **D**ata **L**ength)	1	Длина поля User Data в байтах
UD (**U**ser **D**ata)	0...140	Сообщение

Теперь, для того, чтобы все стало понятно, вместе с подробным описанием каждого из полей PDU-пакета, будет поэтапно показано, как формируется PDU-пакет на примере сообщения с текстом "*Тест формата PDU!*", отправляемого на номер **+7 (890) 123-45-67**.

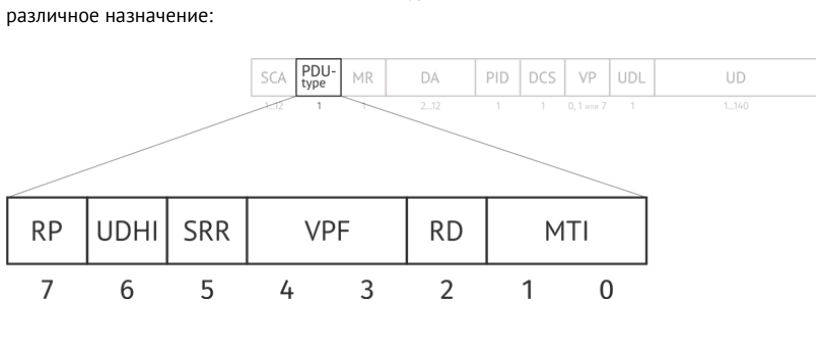
— SCA *(Service Center Address)*

Данное поле является необязательным, так как номер SMS-центра (SMSC) по умолчанию зашит в SIM-карте. При этом значение поля SCA должно быть равно `00h`. Применение дефолтного номера SMSC удобно ещё и тем, что сформированный PDU-пакет не привязан к конкретному мобильному оператору и является универсальным. PDU-пакет:

```
SCA PDU-type
00  ...
```

— PDU-type

Каждый бит этого однобайтового поля **PDU-type** имеет различное назначение:



- **RP** (Reply Path) - путь для ответа:
  `0` — не определен,
  `1` — определен, используется тот же SMSC (центр отправки SMS).
- **UDHI** (User Data Header Included) - содержит ли поле User Data, помимо сообщения, дополнительный заголовок:
  `0` — не содержит,
  `1` — содержит.
- **SRR** (Status Report Request) - запрос статуса сообщения:
  `0` — не запрашивается,
  `1` — запрашивается.
- **VPF** (Validity Period Format) - формат поля VP (Validity Period, время жизни SMS):
  `00` — поле VP отсутствует,
  `01` — зарезервировано,
  `10` — поле VP содержит временные данные в относительном формате,
  `11` — поле VP содержит временные данные в абсолютном формате.
- **RD** (Reject Duplicates) - правила обращения с дубликатами сообщений (при одинаковых полях MR и DA). Используется в случаях, когда в SMS-центр поступает сообщение с такими же значениями полей MR и DA, что и у предыдущего сообщения:
  `0` — переслать сообщение,
  `1` — отклонить сообщение.
- **MTI** (Message Type Indicator) - тип сообщения:
  `00` — принимаемое сообщение или подтверждение приема отправленного SMS,
  `01` — отправляемое сообщение или подтверждение отправки SMS,
  `10` — отчет о доставке SMS или SMS-команда (от модуля к SMSC),
  `11` — зарезервировано.

При использовании значений по умолчанию - все биты нулевые, 2 бита MTI - `01` (тип сообщения - отправляемое SMS), байт **PDU-type** примет значение 1:

RP	UDHI	SRR	VPF		RD	MTI	
0	0	0	0	0	0	0	1

Таким образом, значение байта PDU-type принимается равным `00000001b` = `01h`:

```
SCA PDU-type MR
00  01       ...
```

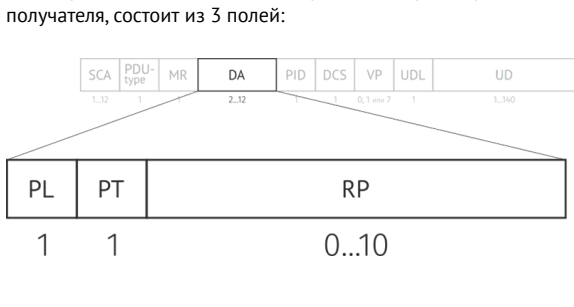
— MR *(Message Reference)*

Автоматически присваиваемый модулем идентификатор (порядковый номер) сообщения. Значение поля MR принимается равным `00h`:

```
SCA PDU-type MR DA
00  01       00 ...
```

— DA *(Destination Address)*

Закодированный специальным образом номер телефона получателя, состоит из 3 полей:



- **PL** (Phone Length) — количество цифр в номере телефона получателя, без учета прочих знаков типа "+". Например для телефона **+7 (897) 012-34-56** количество цифр - 11. Значение поля PL в шестнадцатиричном формате должно быть `11d` = `0Bh`.
- **PT** (Phone Type) — тип номера получателя. В основном используются 2 значения:
  `91h` — международный формат (как в примере +7 (897) 01...),
  `81h` — местный формат, либо короткие номера.
- **RP** (Recipient's Phone) — специально подготовленный номер получателя. Правила подготовки очень простые. Во-первых, нужно оставить только цифры: было `+7 (897) 012-34-56`, стало `78970123456`. Во-вторых, если количество цифр нечетное, то в конце нужно добавить букву `F`: было `78970123456`, стало `78970123456F`. И в-третьих, каждую пару цифр нужно поменять местами: было `78970123456F`, стало `8779103254F6`.

Таким образом поле DA примет вид:

```
PL PT RP
0B 91 8779103254F6
```

PDU-пакет:

```
SCA PDU-type MR DA               PID
00  01       00 0B918779103254F6 ...
```

— PID *(Protocol Identifier)*

Подробнее о значениях поля PID можно прочитать в ETSI GSM 03.40, пункт 9.2.3.9

Идентификатор протокола. Для задач, определенных данной статьей, поле принимается равным `00h`:

```
SCA PDU-type MR DA               PID DCS
00  01       00 0B918779103254F6 00  ...
```

— DCS *(Data Coding Scheme)*

Подробнее о значениях, которые может принимать поле DCS можно почитать здесь W DCS Values

Кодировка сообщения. Поскольку, речь о PDU-формате, идет, исходя из необходимости отправки SMS на русском языке, то данное поле должно устанавливать кодировку UCS2. Для этого можно использовать 2 значения:

- `08h` — для отправки обычных сообщений,
- `18h` — для отправки flash-сообщений.

```
SCA PDU-type MR DA               PID DCS VP
00  01       00 0B918779103254F6 00  08  ...
```

— VP *(Validity Period)*

Время жизни сообщения определяет срок хранения сообщения в SMSC, в случае если его не удается доставить адресату. Это поле связано со значениями битов **VPF** поля **PDU-type**. Подробно об этом будет рассказано далее, а поскольку, в примере заданы нулевые биты **VPF** — `00000001`, то это поле не будет входить в состав PDU-пакета.

```
SCA PDU-type MR DA               PID DCS VP UDL
00  01       00 0B918779103254F6 00  08     ...
```

— UDL *(User Data Length)*

Длина поля UD в байтах. Поскольку каждый символ в кодировке UCS2 кодируется 2 байтами, то для получения значения поля UDL, нужно количество символов в сообщении умножить на 2. Текст примера "*Тест формата PDU!*" состоит из 17 символов (включая пробелы), таким образом значение поля UDL будет представлять `17*2` = `34d` = `22h`:

```
SCA PDU-type MR DA               PID DCS VP UDL UD
00  01       00 0B918779103254F6 00  08     22  ...
```

На заметку: *Поскольку каждый символ кодируется двумя байтами, а максимальная длина сообщения 140 байт, то максимальная длина отправляемого сообщения в PDU-формате может составлять 70 символов (140/2=70).*

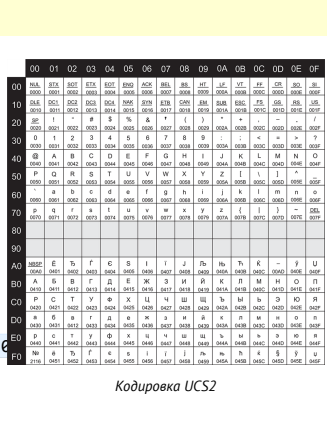
— UD *(User Data)*

Непосредственно сообщение в кодировке UCS. Пример "*Тест формата PDU!*" в кодировке UCS2 будет выглядеть так:

```
Т - 0422        о - 043E          - 0020
е - 0435        р - 0440        P - 0050
с - 0441        м - 043C        D - 0044
т - 0442        а - 0430        U - 0055
  - 0020        т - 0442        ! - 0021
ф - 0444        а - 0430
```

Поле UD — `042204350441044200200444043E0440043C04300442043000200050004400550021`

```
SCA PDU-type MR DA               PID DCS VP UDL UD
00  01       00 0B918779103254F6 00  08     22  042204350441
```



Кодировка UCS2

Сформированный PDU-пакет будет выглядеть так: `0001000B918779103254F6000822042204350441044200200444043E0440043C04300442043000200050004400550021`.

Для того, чтобы отправить SMS в PDU-формате, в первую очередь необходимо установить режим отправки PDU:

```
AT+CMGF=0
```

Далее, командой `AT+CMGS=<length>` в параметре *<length>* необходимо передать длину PDU-пакета в байтах без учета поля **SCA**. Поскольку каждый байт кодируется двумя символами, нужно исключить поле SCA и разделить оставшееся количество символов пополам. Таким образом, значение параметра *<length>* будет равно 47:

```
AT+CMGS=47
```

Далее, как и при отправке SMS в текстовом формате — передается PDU-пакет, и в завершение `Ctrl+Z`.

Отправка SMS в PDU-формате: Arduino

Программная реализация отправки SMS в PDU-формате очень проста, за исключением части, отвечающей за кодирование строки в UCS2-формат. Блок кодирования состоит из нескольких функций и выглядит так:

Arduino (C++)

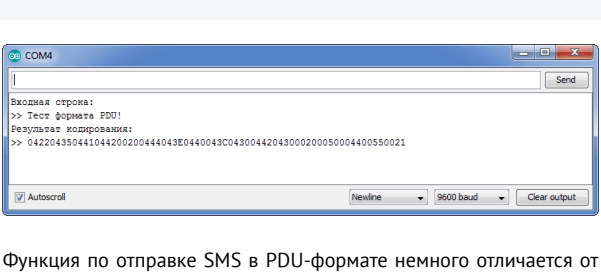
```
// ================================== Блок кодирования строки в представление UCS2
String StringToUCS2(String s)
{
  String output = "";                                               // Переменная дл

  for (int k = 0; k < s.length(); k++) {                            // Начинаем пере
    byte actualChar = (byte)s[k];                                   // Получаем перв
    unsigned int charSize = getCharSize(actualChar);                // Получаем длин

    // Максимальная длина символа в UTF-8 - 6 байт плюс завершающий ноль, итого 7
    char symbolBytes[charSize + 1];                                 // Объявляем мас
    for (int i = 0; i < charSize; i++)  symbolBytes[i] = s[k + i];  // Записываем в
    symbolBytes[charSize] = '\0';                                   // Добавляем зав

    unsigned int charCode = symbolToUInt(symbolBytes);              // Получаем DEC-
    if (charCode > 0)  {                                            // Если все корр
      // Остается каждый из 2 байт перевести в HEX формат, преобразовать в строку и
      output += byteToHexString((charCode & 0xFF00) >> 8) +
                byteToHexString(charCode & 0xFF);
    }
    k += charSize - 1;                                              // Передвигаем у
  }
  return output;                                                    // Возвращаем ре
}

unsigned int getCharSize(unsigned char b) { // Функция получения количества байт, ко
  // По правилам кодирования UTF-8, по старшим битам первого октета вычисляется общи
  // 1  0xxxxxxx - старший бит ноль (ASCII код совпадает с UTF-8) - символ из систем
  // 2  110xxxxx - два старших бита единицы - символ кодируется двумя байтами
  // 3  1110xxxx - 3 байта и т.д.
  // 4  11110xxx
  // 5  111110xx
  // 6  1111110x

  if (b < 128) return 1;             // Если первый байт из системы ASCII, то он код

  // Дальше нужно посчитать сколько единиц в старших битах до первого нуля - таково
  // При помощи маски, поочереди исключаем старшие биты, до тех пор пока не дойдет д
  for (int i = 1; i <= 7; i++) {
    if (((b << i) & 0xFF) >> 7 == 0) {
      return i;
    }
  }
  return 1;
}

unsigned int symbolToUInt(const String& bytes) {  // Функция для получения DEC-предс
  unsigned int charSize = bytes.length();         // Количество байт, которыми закод
  unsigned int result = 0;
  if (charSize == 1) {
    return bytes[0]; // Если символ кодируется одним байтом, сразу отправляем его
  }
```

На заметку: *Для лучшего понимания работы функций кодирования, автор рекомендует ознакомиться с W принципами кодирования UTF-8.*

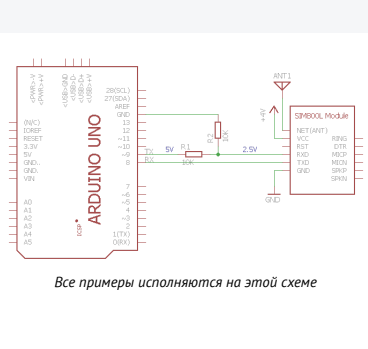
Использование:

Arduino (C++)

```
void setup() {
  Serial.begin(9600);               // Скорость обмена данными с компьютером

  String strTest = "Тест формата PDU!";
  Serial.println("Входная строка:");
  Serial.println(">> " + strTest);
  Serial.println("Результат кодирования:");
  Serial.println(">> " + StringToUCS2(strTest));
}

void loop() {
}
```



```
Входная строка:
>> Тест формата PDU!
Результат кодирования:
>> 042204350441044200200444043E0440043C04300442043000200050004400550021
```

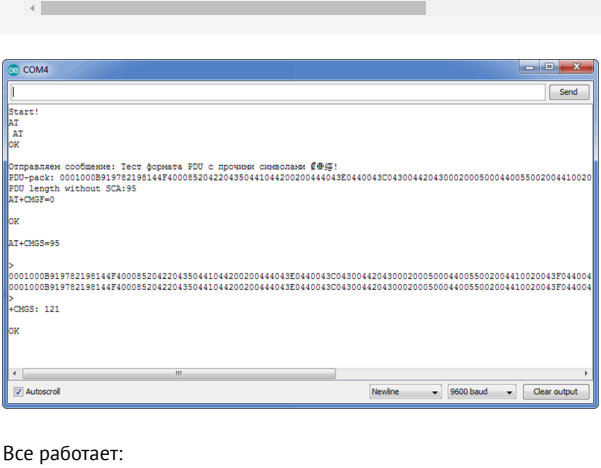


Все примеры исполняются на этой схеме

Функция по отправке SMS в PDU-формате немного отличается от отправки SMS в текстовом формате. Дополнительно вводятся функции по формированию PDU-пакета, по схеме, описанной выше. Полный пример отправки SMS в PDU-формате:

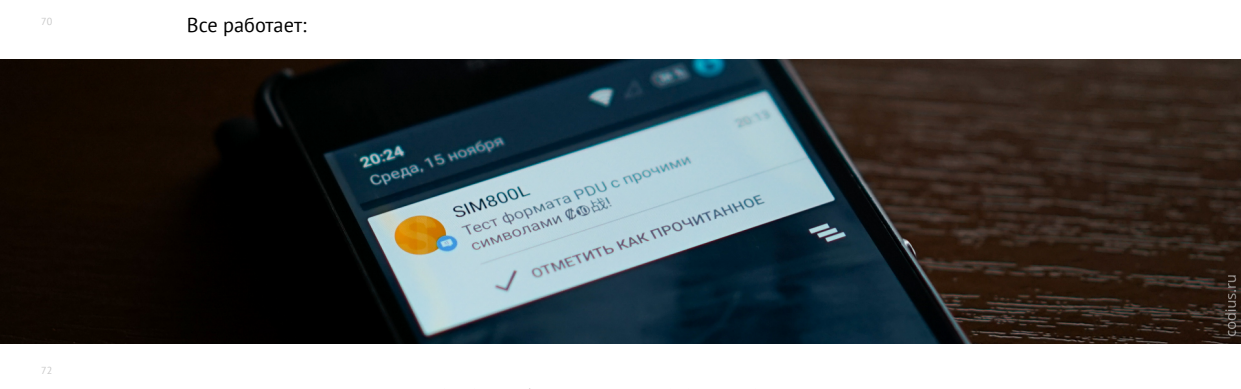
Arduino (C++)

```
#include <SoftwareSerial.h>                          // Библиотека програмной реали
SoftwareSerial SIM800(8, 9);                         // RX, TX

String _response = "";                               // Переменная для хранения отв

void setup() {
  Serial.begin(9600);                                // Скорость обмена данными с к
  SIM800.begin(9600);                                // Скорость обмена данными с м

  Serial.println("Start!");
  sendATCommand("AT", true);                         // Отправили AT для настройки

  String strTest = "Тест формата PDU с прочими символами €®傳!";
  sendSMSinPDU("+7928xxxxxxx", strTest);
}

void loop() {
  if (SIM800.available())   {                        // Если модем, что-то отправил
    _response = waitResponse();                      // Получаем ответ от модема дл
    _response.trim();                                // Убираем лишние пробелы в на
    Serial.println(_response);                       // Если нужно выводим в монито
  }
  if (Serial.available())  {                         // Ожидаем команды по Serial..
    SIM800.write(Serial.read());                     // ...и отправляем полученную
  };
}

void sendSMSinPDU(String phone, String message)
{
  Serial.println("Отправляем сообщение: " + message);

  // ============ Подготовка PDU-пакета =============================================
  // В целях экономии памяти будем использовать указатели и ссылки
  String *ptrphone = &phone;                                // Указатель на пере
  String *ptrmessage = &message;                            // Указатель на пере

  String PDUPack;                                           // Переменная для хр
  String *ptrPDUPack = &PDUPack;                            // Создаем указатель

  int PDUlen = 0;                                           // Переменная для хр
  int *ptrPDUlen = &PDUlen;                                 // Указатель на пере

  getPDUPack(ptrphone, ptrmessage, ptrPDUPack, ptrPDUlen);  // Функция формирующ

  Serial.println("PDU-pack: " + PDUPack);
  Serial.println("PDU length without SCA:" + (String)PDUlen);

  // ============ Отправка PDU-сообщения ============================================
  sendATCommand("AT+CMGF=0", true);                         // Включаем PDU-режи
  sendATCommand("AT+CMGS=" + (String)PDUlen, true);         // Отправляем длину
  sendATCommand(PDUPack + (String)((char)26), true);        // После PDU-пакета
```



```
Start!
AT
 AT
OK

Отправляем сообщение: Тест формата PDU с прочими символами €®傳!
PDU-pack: [illegible]
PDU length without SCA:[illegible]
AT+CMGF=0

OK

AT+CMGS=[illegible]

> [illegible]
> 
+CMGS: 121

OK
```

Все работает:



Установка срока жизни SMS — биты VPF (Validity Period Format) и поле VP (Validity Period)

В примере используемом в статье, в целях упрощения, битам **VPF** было присвоено значение `00`. И здесь стоит напомнить, что эти биты задают формат поля **VP**, могут принимать и другие значения, которые будут влиять на содержимое PDU-пакета, и, как следствие, отношение SMS-центра к отправленному SMS:

- `00` — поле VP отсутствует,
- `01` — зарезервировано,
- `10` — поле VP содержит временные данные в относительном формате,
- `11` — поле VP содержит временные данные в абсолютном формате.

Здесь имеет смысл повторить, что данная информация становится актуальной в случаях, когда нет возможности доставить SMS адресату, когда он, например, находится вне зоны действия сети. И этим параметром можно указать срок, в течение которого SMS-центр не будет его удалять, а будет пытаться его доставить. По истечению этого срока, если сообщение не было доставлено, оно будет удалено.

Далее будут рассмотрены 2 возможных варианта формата поля **VP**, задаваемые битами **VPF**.

Вариант 1 — поле VP содержит данные в относительном формате.

Данный вариант устанавливается значениями битов VPF — `10`. И говорит о том, что поле **VP** хранит данные о сроке жизни SMS в относительном формате, то есть по отношению ко времени его создания. Например, 30 минут, сутки или 30 дней, с момента получения сообщения SMS-центром.

Теперь поле **PDU-type** примет вид `00010001b` = `11h`.

В данном случае, длина поля VP составляет 1 байт. Значение этого байта устанавливается в соответствии с таблицей:

Шестнадцатеричное значение поля VP	Десятичное значение поля VP	Время, соответствующее значению поля VP	Максимальное время жизни
`1...8F`	0...143	(VP + 1) × 5 минут	12 часов
`90...A7`	144...167	12 часов + (VP – 143) × 30 минут	24 часа
`A8...C4`	168...196	(VP – 166) × 1 день	30 дней
`C5...FF`	197...255	(VP – 192) × 1 неделя	63 недели

Например, при необходимом времени жизни сообщения 5 часов, значение будет рассчитано по формуле 1 строки таблицы. Здесь нужно будет решить простое уравнение:

$$(VP+1) \times 5 \text{ минут} = 300 \text{ минут } (5 \text{ часов} \times 60 \text{ минут}),$$
$$(VP+1) = 60,$$
$$VP = 59$$

Таким образом поле VP должно будет принять значение `59d` = `3Bh`.

Исходя из вышеизложенного весь PDU-пакет необходимо пересобрать — изменить поле **PDU-type** и добавить поле **VP**:

```
SCA PDU-type MR DA               PID DCS VP UDL UD
00  11       00 0B918779103254F6 00  08  3B 22  042204350441044
```

Также, необходимо пересчитать, с учетом изменившегося PDU-пакета, его длину, отправляемую параметром команды `AT+CMGS=<length>`.

Вариант 2 — поле VP содержит данные в абсолютном формате.

Данный вариант устанавливается значениями битов VPF — `11`. И говорит о том, что поле **VP** хранит данные о конкретном времени, до наступления которого сообщение не будет удалено. Например, срок жизни сообщения до **09:20:50 30 декабря 2017 года**. При наступлении этого времени, если сообщение не было доставлено, оно удалится.

Поле **PDU-type** теперь будет иметь значение `00011001b` = `19h`.

В данном варианте, длина поля VP составляет 7 байт и его структура выглядит так:

Байт 1	Байт 2	Байт 3	Байт 4	Байт 5	Байт 6	Байт 7
Год	Месяц	День	Час	Минуты	Секунды	Часовой пояс

Каждый байт представлен двухзначным десятичным числом с переставленными цифрами. При обозначении года используются последние две цифры. Часовой пояс показывает смещение времени относительно Гринвича (GMT), выраженную в четвертях часа. В случае отрицательного смещения (GMT-2), третий бит байта устанавливается в `1`. Например, необходимо установить следующее значение **25 марта 2018 года 15:23:54 (GMT-7)**.

```
        Г   М   Д   Ч   м   с   ЧП
было:  18  03  25  15  23  54  -7
стало: 81  30  52  51  32  54  8A
```

Поле VP принимает значение: `8130525132548A`.

Но здесь имеет смысл подробнее остановиться на получении корректного значения часового пояса. Если с положительным смещением, все более-менее просто, то с отрицательным, алгоритм требует пояснений. Ниже представлены несколько примеров получения 7 байта поля VP:

Пример №1. GMT-7 — `8A`
В 7 часах 28 четвертей часа (7×4=28). Меняем цифры местами 28→82. `82h` = `10000010b`. Поскольку значение отрицательное, меняем третий бит на 1:
`10000010b` → `10001010b`. Представление в HEX-формате: `10001010b` → `8Ah`.

Пример №2. GMT-3 — `29`
В 3 часах 12 четвертей часа (3×4=12). Меняем цифры местами 12→21. `21h` = `00100001b`. Поскольку значение отрицательное, меняем третий бит на 1:
`00100001b` → `00101001b`. Представление в HEX-формате: `00101001b` → `29h`.

Пример №3. GMT+4 — `61`
В 4 часах 16 четвертей часа (4×4=16). Меняем цифры местами 16→61. `61h`. Поскольку значение положительное, менять ничего не нужно — `61h`.

Теперь, также как и в предыдущем варианте весь PDU-пакет нужно пересобрать — изменить поле **PDU-type** и добавить поле **VP**:

```
SCA PDU-type MR DA               PID DCS VP             UDL UD
00  19       00 0B918779103254F6 00  08  8130525132548A 22  042
```

И снова нужно будет пересчитать, с учетом изменившегося PDU-пакета, его длину, отправляемую параметром команды `AT+CMGS=<length>`.

← К предыдущей части
К следующей части →

Что почитать:

- *Отправка SMS на русском — Send_Ru_SMS_using_GSM_Neoway.pdf (231 KB)*
- *SMS PDU Mode*
- *SMS and the PDU format*
- *Представление кириллицы в UTF-8*
- *W GSM 03.40*
- *Спецификация GSM 03.40 v5.3.0 (339 KB)*
- *Отправка SMS-сообщений в формате PDU, теория с примерами на C#*

Похожие запросы:

- *Посылаем/принимает SMS на русском*
- *Отправка SMS-сообщений в формате PDU, теория с примерами на Arduino*
- *GSM модуль SIM800L – самое полное руководство на русском языке*
- *Как отправить SMS на русском*
- *Декодирование входящих русских SMS*
- *Декодирование кириллицы из PDU-формата*
- *SMS PDU Mode*
- *Декодирование SMS-сообщений в формате PDU*
- *Как расшифровать PDU-формат*
- *Использование PDU формата для оповещения через SMS сообщения*
- *Формат PDU или как сформировать и распаковать SMS*
- *Что такое PDU и как послать по GSM*

comments powered by HyperComments

© 2013-2024 codius.ru