



АВТОГРАФ серия X. ПРИМЕНЕНИЕ СКРИПТОВ И ОПИСАНИЕ ФУНКЦИЙ

РУКОВОДСТВО ПО ПРИМЕНЕНИЮ

ВЕРСИЯ

1.4



ОГЛАВЛЕНИЕ

Введение	3
Подготовка окружения	4
Компиляция и загрузка микропрограммы	5
Компиляция микропрограмм T.Скрипт через Visual Studio Code	8
Выполнение микропрограмм T.Скрипт в терминале АвтоГРАФ	10
Документация	10
Функции для работы с CAN-шиной	11
Функции для работы с текстовыми командами	16
Функции для работы с событиями	22
Функции для работы со входами	26
Различные встроенные функции	32
Функции для работы с навигационными приемниками	43
Функции для работы с параметрами устройств АвтоГРАФ	46
Функции для работы с последовательными портами	93
Функции для работы с сокетом	99
Функции для работы с файлами в SPI памяти	104
Функции для работы с файлами в SD/RAM памяти	114
Функции для работы с BLE	124
Стандартные функции Rawp	133
Стандартные функции Rawp для чисел с фиксированной запятой	147
Стандартные функции Rawp для чисел с плавающей запятой	157
Стандартные функции Rawp для строк	174
Стандартные функции Rawp для времени и таймера	192

Введение

В терминалах АвтоГРАФ-GX реализована технология исполнения микропрограмм, написанных пользователем, — Т.Скрипт:

- в АвтоГРАФ-LX — начиная с серийного номера **2909800**;
- в АвтоГРАФ-LX (E) — начиная с серийного номера **2823000**;
- в АвтоГРАФ-SX — начиная с серийного номера **2478000**;
- в АвтоГРАФ-GX — начиная с серийного номера **3115000**;
- в АвтоГРАФ-GX Wi-Fi — начиная с серийного номера **3029000**;
- в АвтоГРАФ-GX АКБ — начиная с серийного номера **3996500**;
- в АвтоГРАФ-GX Wi-Fi АКБ — начиная с серийного номера **3990200**.

В качестве языка программирования используется компилируемый язык Pawn (<https://www.compuphase.com/pawn/pawn.htm>). Написание исходного кода и компиляция выполняются в среде разработки Pawn IDE.

Использование микропрограмм Т.Скрипт позволяет:

- получить доступ к интерфейсам терминала АвтоГРАФ;
- формировать собственные телеметрические данные;
- управлять формированием стандартных телеметрических данных терминала;
- получить доступ к параметрам терминала (например, CAN параметры, навигационные параметры, дискретные параметры, флаги устройства и прочее);
- выполнять команды удаленной настройки (SMS- и серверные команды) непосредственно из микропрограммы.

Выполнение микропрограмм Т.Скрипт защищено от несанкционированного доступа к настройкам терминала: при установленном уровне защиты 1 скрипты терминала недоступны для изменения, а при установленном уровне защиты 2 — недоступны для считывания. Также выполнение микропрограмм Т.Скрипт защищено от несанкционированного доступа к памяти терминала. Это позволяет сохранить надежность работы основной микропрограммы терминала на высоком уровне. Подробнее о выполнении микропрограмм см. раздел [«Выполнение микропрограмм Т.Скрипт в терминале АвтоГРАФ»](#).

Подготовка окружения

Для создания и компиляции микропрограмм Т.Скрипт нужно подготовить окружение:

1. Загрузите архив с файлами скриптов *.inc* по ссылке https://i.tk-chel.ru/products/T_SCRIPT/t.script.zip и распакуйте его содержимое в любую директорию. Архив также содержит примеры скриптов, разработанных специалистами компании «ТехноКом».
2. Загрузите установочный файл для Pawn IDE по ссылке https://i.tk-chel.ru/products/T_SCRIPT/pawn-4.1.7152.exe.
3. Запустите загруженный файл *pawn-4.1.7152.exe* и выполните установку в любую директорию.
4. Запустите Pawn IDE (Quincy) через меню «Пуск» (каталог «Pawn»).
5. Выберите в главном меню Pawn IDE пункт «Tools», далее — пункт «Options...» (**Рис.1**) и укажите в поле «Include path» путь к файлам скриптов *.inc* от ТехноКом (**Рис.2**).

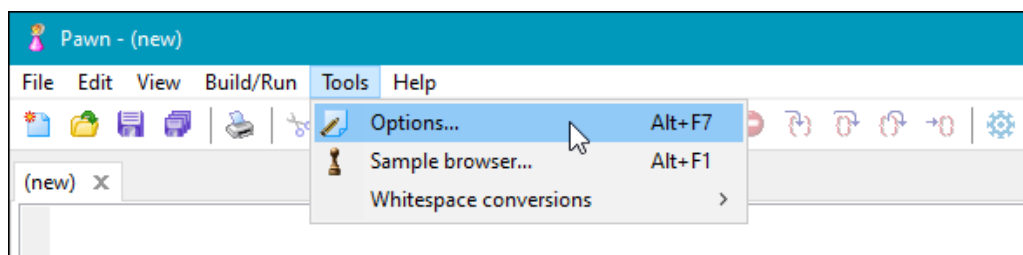


Рис.1. Переход к настройкам Pawn IDE

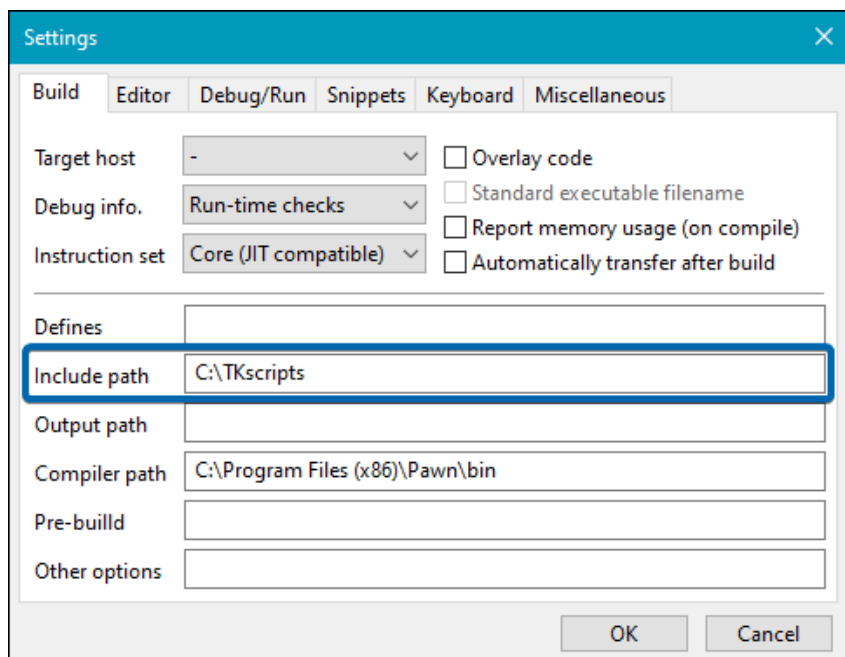


Рис.2. Указание пути к файлам скриптов от ТехноКом

Компиляция и загрузка микропрограммы

Пример использования возможностей микропрограмм Т.Скрипт, содержащийся в файле *tkExample.p*, демонстрирует следующие возможности:

- периодическую выдачу в отладку сообщений в debug интерфейс CDC и файлов логов терминала;
- формирование и сохранение пользовательской телеметрической информации;
- работу с интерфейсом RS232 в виде реализации функции ЭХО.

Для **компиляции и загрузки** микропрограммы в терминал выполните следующий порядок действий:

1. Запустите Pawn IDE.
2. Откройте файл с исходным текстом скрипта *tkExample.p* (Рис.3).

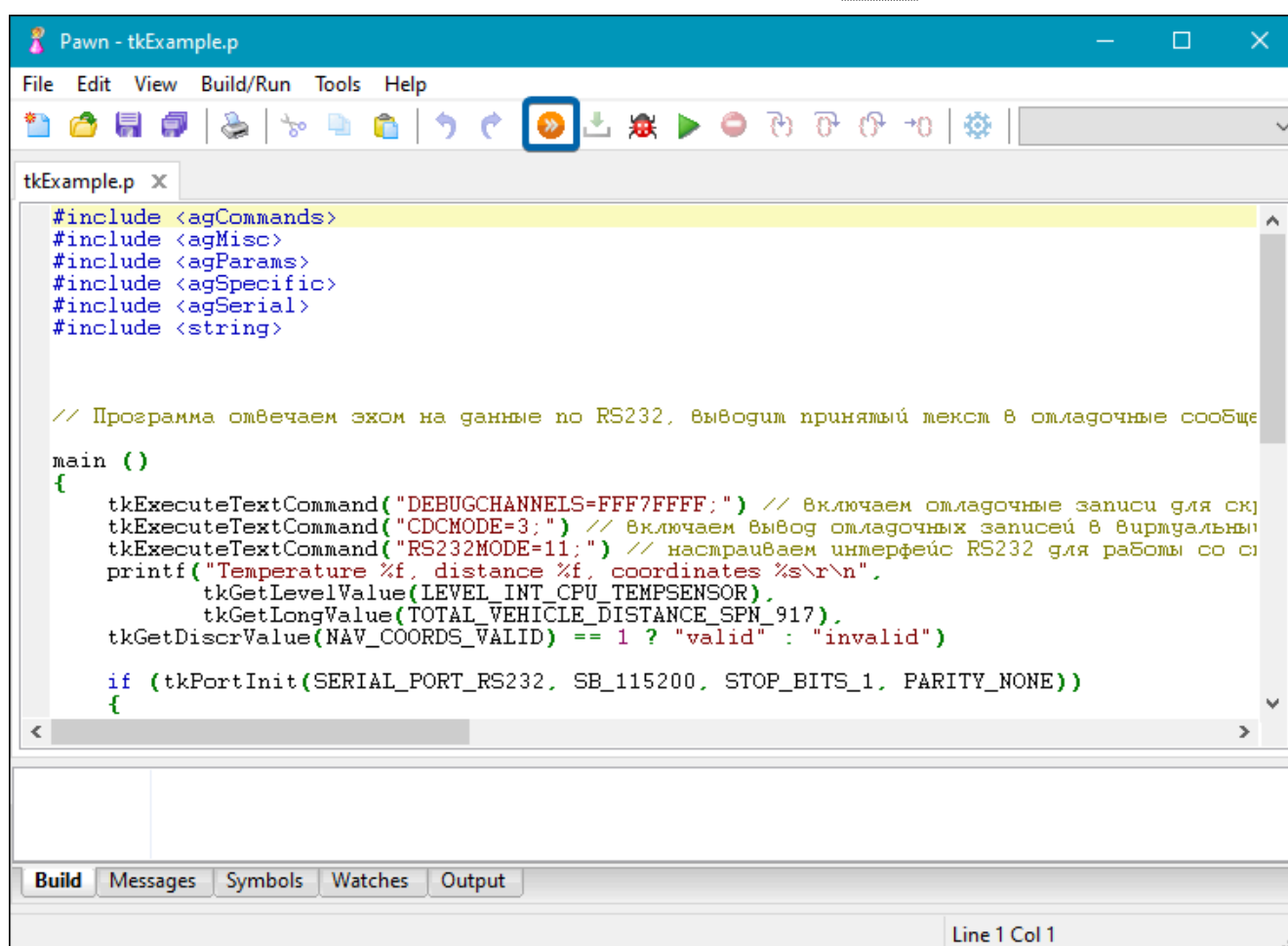


Рис.3. Исходный текст скрипта

3. Нажмите на клавиатуре клавишу F7 или кнопку «Compile Current Script» на панели инструментов Pawn IDE — в директории с файлом *tkExample.p* появится файл *tkExample.amx* (Рис.4).

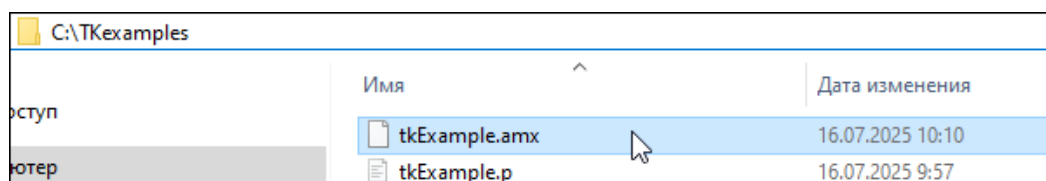


Рис.4. Скомпилированный файл

4. Скопируйте файл `tkExample.amx` в директорию `MCU\SCRIPTS` в терминале. Для этого воспользуйтесь файловым менеджером в конфигураторе АвтоГРАФ GSMConf 5.0 или отправьте файл через сервер.
5. Включите в терминале запись скриптов в журнал с помощью соответствующего чек-бокса в разделе «Логи» конфигуратора или с помощью команды `DEBUGCHANNELS=FFF7FFF;`
6. Разрешите работу скриптов в терминале с помощью переключателя в конфигураторе или команды `SCRIPTSENABLE=1;`
7. Проверьте работу скрипта через отладочный интерфейс CDC конфигуратора, выбрав режим «Вывод отладочной информации» (Рис.5).

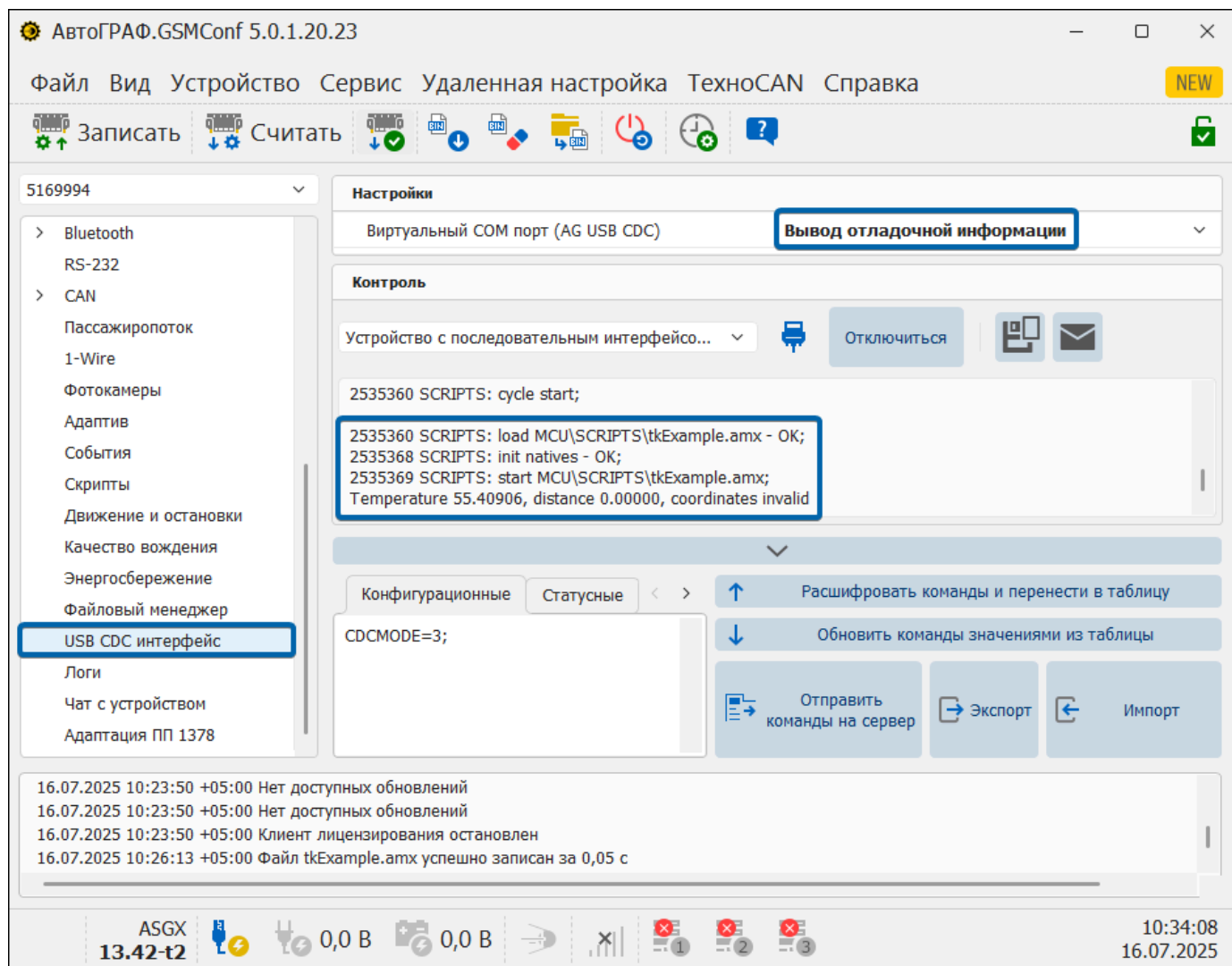


Рис.5. Проверка работы скрипта через отладочный интерфейс CDC

8. Удаленно проверьте наличие файла со скриптом с помощью команды `DIRTREE;`

Процесс выполнения можно проследить в лог-файлах терминала на сервере, если для терминала в конфигураторе включена опция «Передавать логи на сервер» (Рис.6).

Для **обновления** микропрограммы в терминале загрузите в него обновленный файл `.amx`. Если имя файла, имеющегося в терминале, совпадает с именем загружаемого файла, то файл в терминале будет перезаписан автоматически. Для исполнения перезаписанного файла `.amx` нужно отправить команду, которая перезапускает работу скриптов в терминале, — `SCRIPTSRESTART;`

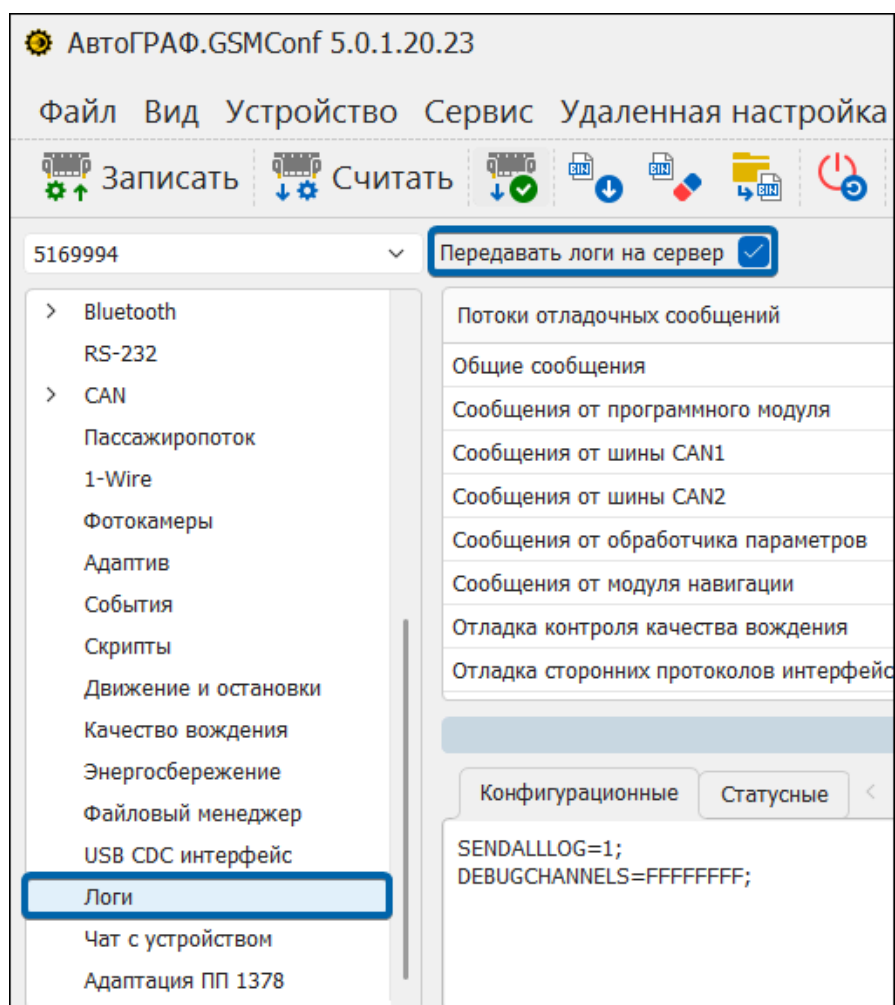


Рис.6. Включение передачи логов на сервер

Для **удаления** файлов со скриптами из терминала отправьте команду `FORMATMCU`. По этой команде все файлы в директории `MCU` будут удалены. Удаление файлов по отдельности не предусмотрено.

Компиляция микропрограмм T.Скрипт через Visual Studio Code

Такой способ компиляции микропрограмм T.Скрипт позволяет:

- проводить контроль целостности файлов загружаемых микропрограмм (путем автоматического запуска файла `amx_to_agx.exe` после компиляции);
- пользоваться расширением для подсветки синтаксиса в Visual Studio Code (например, Pawn Tools);
- пользоваться AI-ассистентом (например, GitHub Copilot).

Предварительно требуется загрузить архив с файлами для компиляции через Visual Studio Code по ссылке https://i.tk-chel.ru/products/T_SCRIPT/Pawn_VSCode.zip.

Для компиляции микропрограммы выполните следующий порядок действий:

1. Скопируйте папку `.vscode` в директорию с файлом скрипта.
2. Скопируйте файл `amx_to_agx.exe` в любую директорию (можно в папку с файлом скрипта).
3. Перейдите в скопированную папку `.vscode` и отредактируйте в ней файл `task.json` (Рис.7):
 - замените значение в строке `"command": "C:\\example\\pawnc.exe"` на путь до вашего компилятора;
 - замените значение в строке `"-iC:\\scripts\\inc"` на путь до папки с файлами скриптов `.inc`;
 - замените значение в строке `"command": "C:\\example\\amx_to_agx.exe"` на путь до скопированного файла `amx_to_agx.exe`.

```
1 {
2   "version": "2.0.0",
3   "tasks": [
4     {
5       "label": "build-normal",
6       "type": "shell",
7       "command": "C:\\example\\pawnc.exe",
8       "args": [
9         "${fileBasename}",
10        "-D${fileDirname}",
11        "-iC:\\scripts\\inc",
12        "-(+'"
13      ],
14      "presentation": {
15        "reveal": "always",
16        "panel": "shared",
17        "clear": true
18      },
19      "problemMatcher": [],
20      "group": "none"
21    },
22    {
23      "label": "convert-amx",
24      "type": "shell",
25      "command": "C:\\example\\amx_to_agx.exe",
26      "args": [
27        "${fileDirname}\\${fileBasenameNoExtension}.amx"
28      ],
```

Рис.7. Редактирование файла `task.json`

4. Если требуется, чтобы вместо файла `.agx` собирался файл `.amx`, то удалите строку `"convert-amx"` (**Рис.8**). Следует учитывать, что в таком случае не будет гарантии целостности собранного файла.

```
37 {
38   "label": "build-and-convert",
39   "dependsOrder": "sequence",
40   "dependsOn": [
41     "build-normal",
42     "convert-amx"
43   ],
```

Рис.8. Настройка сборки файла `.amx`

5. Запустите программу Visual Studio Code и откройте в ней папку с файлом скрипта.
6. Откройте файл скрипта, который нужно скомпилировать, и выполните сборку, выбрав в меню программы пункты `Terminal | Run Build Task...` или нажав на клавиатуре комбинацию клавиш `Ctrl+Shift+B`.

Примечание. Для удобства компиляции рекомендуется изменить комбинацию клавиш сборки (например, на `F5`): выберите в меню программы пункты `File | Preferences | Keyboard Shortcuts`, найдите команду «`Tasks: Run Build Task`» и назначьте ей нужную клавишу или комбинацию клавиш.

Выполнение микропрограмм Т.Скрипт в терминале АвтоГРАФ

Микропрограммы Т.Скрипт выполняются в отдельной задаче операционной системы реального времени (RTOS). Приоритет задачи Т.Скрипт ниже, чем у задач, выполняющих базовые функции прошивки АвтоГРАФ. Используемая RTOS вытесняющего типа работает по следующему принципу: если в процессе выполнения задачи Т.Скрипт появится задача с более высоким приоритетом в состоянии «готова к исполнению», то исполнение переключится на эту более приоритетную задачу.

В ПЗУ и ОЗУ терминалов АвтоГРАФ-GX (ASGX) под нужды выполнения микропрограмм Т.Скрипт выделен объем памяти в 1 Мбайт.

Документация

Настоящий документ сформирован на основании прошивки **13.46-a1**.

Оригинальная документация по языку программирования Pawn доступна по ссылке <https://www.compuphase.com/pawn/pawn.htm> или в директории с установленной Pawn IDE (...\\Pawn\\doc).

К ознакомлению рекомендованы следующие файлы:

- Pawn_Getting_Started.pdf — общее описание языка.
- Pawn_Language_Guide.pdf — подробное руководство по языку Pawn.
- Floating_Point_Support — описание библиотеки работы с числами с плавающей запятой.
- Fixed_Point_Support.pdf — описание библиотеки работы с числами с фиксированной запятой.
- String_Manipulation.pdf — описание библиотеки работы со строками.
- Time_Functions.pdf — описание библиотеки работы с временем и таймером.
- Arguments_Support.pdf — описание библиотеки работы с аргументами.

Списки и описание функций, реализованных в терминале АвтоГРАФ, приведены в последующих разделах.

Функции для работы с CAN-шиной

Список функций	Описание
tkCanInit	Инициализация CAN-шины.
tkCanSetFilter	Установка фильтра приема.
tkCanSend	Отправка сообщения в CAN-шину.
tkCanReceive	Прием сообщения из CAN-шины.

tkCanInit

Инициализация CAN-шины.

Для использования функции подключите файл:

```
#include <agCan.inc>
```

Формат функции:

tkCanInit(speed, active, port)

Параметры:

speed	Скорость шины (по умолчанию 250 кбит/с).
active	Разрешить отправку АСК в шину: <ul style="list-style-type: none">• true — разрешить;• false — запретить.
port	Номер CAN-шины: <ul style="list-style-type: none">• 0 — CAN_1 (по умолчанию);• 1 — CAN_2.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkCanInit(500000, true, CAN_1)
```

tkCanSetFilter

Установка фильтра приема.

Для использования функции подключите файл:

```
#include <agCan.inc>
```

Формат функции:

tkCanSetFilter(id, mask, type, num_filter, port)

Параметры:

id	Идентификатор CAN-сообщения.
mask	Маска CAN-сообщения.
type	Тип используемых идентификаторов: <ul style="list-style-type: none">• 0 — STD (11-битные);• 1 — EXT (29-битные).
num_filter	Номер фильтра (0–31).
port	Номер CAN-шины: <ul style="list-style-type: none">• 0 — CAN_1 (по умолчанию);• 1 — CAN_2.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkCanSetFilter(0x100, 0x700, 0, CAN_1);
```

tkCanSend

Отправка сообщения в CAN-шину.

Для использования функции подключите файл:

```
#include <agCan.inc>
```

Формат функции:

tkCanSend(CANMSG, port)

Параметры:

CANMSG[]	Сообщение, отправляемое в CAN-шину.
port	Номер CAN-шины: <ul style="list-style-type: none">• 0 — CAN_1 (по умолчанию);• 1 — CAN_2.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
new canMessage[CANMSG];  
canMessage.id = 0x18DAEEF1;  
canMessage.idType = 1;  
canMessage.dataSize = 8;  
canMessage.data[0] = 0x02107E00;  
tkCanSend(canMessage);
```

tkCanReceive

Прием сообщения из CAN-шины.

Для использования функции подключите файл:

```
#include <agCan.inc>
```

Формат функции:

tkCanReceive(CANMSG, port)

Параметры:

CANMSG[]	Сообщение, получаемое из CAN-шины.
port	Номер CAN-шины: <ul style="list-style-type: none">• 0 — CAN_1 (по умолчанию);• 1 — CAN_2.

Возвращаемое значение:

Принято новое сообщение (1 — сообщение получено, 0 — сообщение не получено).

Пример вызова функции:

```
new can_message[CANMSG];  
tkCanReceive(can_message)
```

Функции для работы с текстовыми командами

Список функций	Описание
tkExecuteTextCommand	Выполнение текстовой команды.
tkReadScriptConfig	Чтение пользовательских настроек.
tkWriteScriptConfig	Запись пользовательских настроек.
tkReadScriptConfigSigned	Чтение пользовательских настроек (со знаком).
tkWriteScriptConfigSigned	Запись пользовательских настроек (со знаком).

tkExecuteTextCommand

Выполнение текстовой команды.

Для использования функции подключите файл:

```
#include <agCommands.inc>
```

Формат функции:

```
tkExecuteTextCommand(const command[], answer[], maxlength=sizeof answer)
```

Параметры:

command	Текст исполняемой текстовой команды.
answer	Буфер под ответ команды.
maxlength	Размер буфера под ответ команды.

Возвращаемое значение:

true — команда была выполнена, false — команда не была выполнена.

Примечание. Если текст ответа не требуется, то параметр **answer** можно не передавать.

Примечание. Отправленные этой функцией настройки автоматически не сохраняются в энергонезависимой памяти и могут быть потеряны при отключении питания устройства. Для сохранения настроек отправьте текстовую команду **SAVECONF** после завершения конфигурирования либо перезагрузите устройство.

Примечание. Чтобы избежать исчерпания ресурсов перезаписи внутренней флеш-памяти, вызов команды **SAVECONF** из Т.Скрипт ограничен по времени: не чаще 10 раз в минуту, 20 раз в час и 30 раз в день.

Пример вызова функции:

```
var answer[128];
if(tkExecuteTextCommand("GVERSION", answer)
{
    printf("Version %s\r\n", answer)
}
```

tkReadScriptConfig

Чтение пользовательских настроек.

Для использования функции подключите файл:

```
#include <agCommands.inc>
```

Формат функции:

tkReadScriptConfig(configId)

Параметры:

configId	Порядковый номер пользовательской настройки, 1...10.
-----------------	--

Возвращаемое значение:

Значение пользовательской настройки.

Примечание. Значение пользовательской настройки может быть записано функцией `tkWriteScriptConfig` или текстовой командой `SCRIPTCONFIGn`.

Пример вызова функции:

```
printf("Config 1 value %d", tkReadScriptConfig(1))
```

tkWriteScriptConfig

Запись пользовательских настроек.

Для использования функции подключите файл:

```
#include <agCommands.inc>
```

Формат функции:

tkWriteScriptConfig(configId, value)

Параметры:

configId	Порядковый номер пользовательской настройки, 1...10.
value	Значение пользовательской настройки, 0...4294967295 (0xFFFFFFFF).

Возвращаемое значение:

Нет.

Примечание. Значение пользовательской настройки может быть считано функцией `tkReadScriptConfig` или текстовой командой `GSCRIPTCONFIGn`.

Примечание. Отправленные этой функцией настройки автоматически не сохраняются в энергонезависимой памяти и могут быть потеряны при отключении питания устройства. Для сохранения настроек отправьте текстовую команду `SAVECONF` после завершения конфигурирования либо перезагрузите устройство.

Внимание! Не рекомендуется часто (чаще одного раза в сутки) менять значение пользовательских настроек. Частая смена значения может привести к исчерпанию ресурсов перезаписи внутренней флеш-памяти.

Примечание. Чтобы избежать исчерпания ресурсов перезаписи внутренней флеш-памяти, вызов команды `SAVECONF` из Т.Скрипт ограничен по времени: не чаще 10 раз в минуту, 20 раз в час и 30 раз в день.

Пример вызова функции:

```
tkWriteScriptConfig(1,100)
```

tkReadScriptConfigSigned

Чтение пользовательских настроек (со знаком).

Для использования функции подключите файл:

```
#include <agCommands.inc>
```

Формат функции:

tkReadScriptConfigSigned(configId)

Параметры:

configId	Порядковый номер пользовательской настройки, 1...10.
-----------------	--

Возвращаемое значение:

Знаковое 32-битное значение пользовательской настройки.

Примечание. Значение пользовательской настройки может быть записано функцией `tkWriteScriptConfigSigned` или текстовой командой `SCRIPTCONFIGnS` (если реализована запись знаковых значений).

Пример вызова функции:

```
printf("Config 1 value %d", tkReadScriptConfigSigned(1));
```

tkWriteScriptConfigSigned

Запись пользовательских настроек (со знаком).

Для использования функции подключите файл:

```
#include <agCommands.inc>
```

Формат функции:

tkWriteScriptConfigSigned(configId, value)

Параметры:

configId	Порядковый номер пользовательской настройки, 1...10.
value	Знаковое 32-битное значение пользовательской настройки, -2147483648...2147483647.

Возвращаемое значение:

Нет.

Примечание. Отправленные этой функцией настройки автоматически не сохраняются в энергонезависимой памяти и могут быть потеряны при отключении питания устройства. Для сохранения настроек отправьте текстовую команду `SAVECONF` после завершения конфигурирования либо перезагрузите устройство.

Внимание! Не рекомендуется часто (чаще одного раза в сутки) менять значение пользовательских настроек. Частая смена значения может привести к исчерпанию ресурсов перезаписи внутренней флеш-памяти.

Примечание. Чтобы избежать исчерпания ресурсов перезаписи внутренней флеш-памяти, вызов команды `SAVECONF` из Т.Скрипт ограничен по времени: не чаще 10 раз в минуту, 20 раз в час и 30 раз в день.

Пример вызова функции:

```
tkWriteScriptConfigSigned(1, -100);
```

Функции для работы с событиями

Список функций	Описание
<u>tkGetEventState</u>	Получение состояния события.
<u>tkGetEconomyState</u>	Получение состояния режима экономии.
<u>tickUntilReset</u>	Получение количества миллисекунд до плановой перезагрузки системы.

tkGetEventState

Получение состояния события.

Для использования функции подключите файл:

```
#include <agEvents.inc>
```

Формат функции:

```
tkGetEventState(eventNumber, eventData[])
```

Параметры:

eventNumber	Порядковый номер события, 1...16.
eventData	Массив структур, куда записываются данные о событии: <ul style="list-style-type: none">• .eventState — состояние события:<ul style="list-style-type: none">• 1 — сработка;• 0 — нет сработки.• .eventTrigger — ожидается действие по сработке:<ul style="list-style-type: none">• 1 — ожидается;• 0 — не ожидается.• .eventDeTrigger — ожидается действие по окончании события:<ul style="list-style-type: none">• 1 — ожидается;• 0 — не ожидается.• .sourceState — текущее состояние источника события (зависит от управляющих команд EVENTSOURCE и EVENTTYPE).• .eventTime — время нахождения в состоянии срабатывания, в миллисекундах.

Возвращаемое значение:

true — событие находится в состоянии сработки, false — событие не находится в состоянии сработки.

Примечание. Если расширенная информация о событии не требуется, то параметр **eventData** можно не передавать.

Пример вызова функции:

```
new eventData[.eventState, .eventTrigger, .eventDeTrigger, .sourceState, .eventTime]
tkGetEventState(1, eventData)
printf("Event 1 state %d, time %d\r\n", eventData.eventState, eventData.eventTime)
printf("Event 2 state %d\r\n", tkGetEventState(2))
```

tkGetEconomyState

Получение состояния режима экономии.

Для использования функции подключите файл:

```
#include <agEvents.inc>
```

Формат функции:

```
tkGetEconomyState(eventData[])
```

Параметры:

eventData	<p>Массив структур, куда записываются данные о режиме:</p> <ul style="list-style-type: none">• .eventState — состояние события:<ul style="list-style-type: none">• 1 — сработка;• 0 — нет сработки.• .eventTrigger — ожидается действие по сработке:<ul style="list-style-type: none">• 1 — ожидается;• 0 — не ожидается.• .eventDeTrigger — ожидается действие по окончании события:<ul style="list-style-type: none">• 1 — ожидается;• 0 — не ожидается.• .sourceState — текущее состояние источника события (зависит от управляющих команд ECONOMYSOURCE и ECONOMYTYPE).• .eventTime — время нахождения в состоянии экономии, в миллисекундах.
------------------	--

Возвращаемое значение:

true — устройство находится в режиме экономии, false — устройство не находится в режиме экономии.

Примечание. Если расширенная информация об экономии не требуется, то параметр **eventData** можно не передавать.

Пример вызова функции:

```
new economyData[.eventState, .eventTrigger, .eventDeTrigger, .sourceState, .eventTime]
tkGetEconomyState(economyData)
printf("Economy state %d, time %d\r\n", economyData.eventState, economyData.eventTime)
printf("Economy state %d\r\n", tkGetEconomyState())
```

tickUntilReset

Получение количества миллисекунд до плановой перезагрузки системы.

Для использования функции подключите файл:

```
#include <agEvents.inc>
```

Формат функции:

```
tickUntilReset ()
```

Параметры:

Нет.

Возвращаемое значение:

Время до следующей плановой перезагрузки системы, в миллисекундах.

Пример вызова функции:

```
if ( tickUntilReset() > 1000 ) {  
    // Работаем, если есть время до перезагрузки  
}
```

Функции для работы со входами

Список функций	Описание
tkGetInputDiscr	Получение состояния входа.
tkGetInputAnalog	Получение аналогового значения входа.
tkGetInputFrequency	Получение частоты цифрового импульсного сигнала на входе.
tkGetInputCounter	Получение счетчика импульсов на входе.
tkSetOutputValue	Установка значения на выходе.

tkGetInputDiscr

Получение состояния входа.

Для использования функции подключите файл:

```
#include <agInputs.inc>
```

Формат функции:

tkGetInputDiscr(input_name)

Параметры:

input_name	Номер входа. Может принимать значения, описанные константой input_name: <ul style="list-style-type: none">• 0 — INPUT_U_1;• 1 — INPUT_U_2;• 2 — INPUT_U_3;• 3 — INPUT_U_4;• 4 — INPUT_U_5;• 5 — INPUT_U_6;• 6 — INPUT_U_7;• 7 — INPUT_U_8.
-------------------	---

Возвращаемое значение:

true — на входе лог «1», false — на входе лог «0».

Пример вызова функции:

```
new bool: value = tkGetInputDiscr(INPUT_U_1);
```

tkGetInputAnalog

Получение аналогового значения входа.

Для использования функции подключите файл:

```
#include <agInputs.inc>
```

Формат функции:

tkGetInputAnalog(input_name)

Параметры:

input_name	Номер входа. Может принимать значения, описанные константой input_name: <ul style="list-style-type: none">• 0 — INPUT_U_1;• 1 — INPUT_U_2;• 2 — INPUT_U_3;• 3 — INPUT_U_4;• 4 — INPUT_U_5;• 5 — INPUT_U_6;• 6 — INPUT_U_7;• 7 — INPUT_U_8.
-------------------	---

Возвращаемое значение:

Напряжение на входе, в милливольтках.

Пример вызова функции:

```
new value = tkGetInputAnalog(INPUT_U_1);
```

tkGetInputFrequency

Получение частоты цифрового импульсного сигнала на входе.

Для использования функции подключите файл:

```
#include <agInputs.inc>
```

Формат функции:

tkGetInputFrequency(input_name)

Параметры:

input_name	Номер входа. Может принимать значения, описанные константой input_name: <ul style="list-style-type: none">• 0 — INPUT_U_1;• 1 — INPUT_U_2;• 2 — INPUT_U_3;• 3 — INPUT_U_4;• 4 — INPUT_U_5;• 5 — INPUT_U_6;• 6 — INPUT_U_7;• 7 — INPUT_U_8.
-------------------	---

Возвращаемое значение:

Частота сигнала (Float), в герцах.

Пример вызова функции:

```
new Float: freq = tkGetInputFrequency(INPUT_U_2);
```

tkGetInputCounter

Получение счетчика импульсов на входе.

Для использования функции подключите файл:

```
#include <agInputs.inc>
```

Формат функции:

tkGetInputCounter(input_name)

Параметры:

input_name	Номер входа. Может принимать значения, описанные константой input_name: <ul style="list-style-type: none">• 0 — INPUT_U_1;• 1 — INPUT_U_2;• 2 — INPUT_U_3;• 3 — INPUT_U_4;• 4 — INPUT_U_5;• 5 — INPUT_U_6;• 6 — INPUT_U_7;• 7 — INPUT_U_8.
-------------------	---

Возвращаемое значение:

Количество импульсов, накопленных на входе с момента последнего обнуления.

Пример вызова функции:

```
new count = tkGetInputCounter(INPUT_U_3);
```

tkSetOutputValue

Установка значения на выходе.

Для использования функции подключите файл:

```
#include <agInputs.inc>
```

Формат функции:

tkSetOutputValue(outputNum, value)

Параметры:

outputNum	Номер выхода. Может принимать значения, описанные константой output_name: <ul style="list-style-type: none">• 0 — OUTPUT_1;• 1 — OUTPUT_2;• 2 — OUTPUT_3.
value	Новое состояние выхода: <ul style="list-style-type: none">• true — выход замкнут (притянут к земле);• false — выход разомкнут.

Пример вызова функции:

```
tkSetOutputValue(OUTPUT_1, false);
```

Различные встроенные функции

Список функций	Описание
tkSwapBuf	Изменение порядка байтов в ячейках массива.
tkGetIntFromBuf	Получение числа (4-байтового int) из буфера по позиции.
tkGetIMEI	Получение IMEI устройства и сохранение в массив.
printfHex	Вывод массива в шестнадцатеричном формате.
tkCrc8	Вычисление CRC-8/MAXIM.
tkCrc16	Вычисление CRC-16.
tkCrc32	Вычисление CRC-32/JAMCRC.
tkCrc8_MinTrans	Вычисление CRC-8.
tkCrc16_MinTrans	Вычисление CRC-16/CCITT.
memset	Заполнение указанной области памяти заданным значением.

tkSwapBuf

Изменение порядка байтов в ячейках массива.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

Функция позволяет поменять байты местами внутри ячейки массива из big-endian в little-endian.

Формат функции:

tkSwapBuf(buf[], size)

Параметры:

buf[]	Имя модифицируемого массива.
size	Размер модифицируемого массива, должен быть кратен 4.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkSwapBuf(buf, 16)
```

tkGetIntFromBuf

Получение числа (4-байтового int) из буфера по позиции.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

Формат функции:

tkGetIntFromBuf(buf{}, index)

Параметры:

buf	Буфер, из которого будет извлекаться число.
index	Индекс байта, начиная с которого число располагается в массиве.

Возвращаемое значение:

Число, извлеченное из буфера.

Пример вызова функции:

```
new buf{15} = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};  
new index = 2;  
new ret = tkGetIntFromBuf(buf, index);
```

tkGetIMEI

Получение IMEI устройства и сохранение в массив.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

Формат функции:

tkGetIMEI(res[], resSize)

Параметры:

res	Массив, в который будет записан IMEI в виде строки символов.
resSize	Размер массива res (должен быть не менее 16).

Возвращаемое значение:

Нет.

Пример вызова функции:

```
new imei[16];  
tkGetIMEI(imei, sizeof imei);  
printf("imei: %s\r\n", imei);
```

printfHex

Вывод массива в шестнадцатеричном формате.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

Формат функции:

```
printfHex(buf[], size)
```

Параметры:

buf	Массив байтов для вывода в шестнадцатеричном формате.
size	Размер массива в байтах.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
new data{5} = { 0xDE, 0xAD, 0xBE, 0xEF, 0x01 };  
printfHex(data, 5);
```

tkCrc8

Вычисление CRC-8/MAXIM.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

- Poly : $0x31 \ x^8 + x^5 + x^4 + 1$
- Init : 0x00
- Revert : true
- XorOut : 0x00
- Check : 0xA1 («123456789»)

Формат функции:

tkCrc8(crc, buf[], len)

Параметры:

crc	Начальное значение CRC.
buf	Массив байтов, для которого вычисляется CRC.
len	Количество байтов в массиве buf .

Возвращаемое значение:

CRC-8 значение.

Пример вызова функции:

```
new data{9} = "123456789";  
new crc_init = 0x00;  
new crc = tkCrc8(crc_init, data, 9);  
// Ожидаемый результат: 0xA1
```

tkCrc16

Вычисление CRC-16.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

- Poly : $0x8005 x^{16} + x^{15} + x^2 + 1$
- Init : 0xFFFF
- Revert : true
- XorOut : 0x0000
- Check : 0x4B37 («123456789»)

Формат функции:

tkCrc16(crc, buf[], len)

Параметры:

crc	Начальное значение CRC.
buf	Массив байтов, для которого вычисляется CRC.
len	Количество байтов в массиве buf .

Возвращаемое значение:

CRC-16 значение.

Пример вызова функции:

```
new data{9} = "123456789";  
new crc_init = 0xFFFF;  
new crc = tkCrc16(crc_init, data, 9);  
// Ожидаемый результат: 0x4B37
```

tkCrc32

Вычисление CRC-32/JAMCRC.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

- Poly : 0x04C11DB7 $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- Init : 0xFFFFFFFF
- Revert : true
- XorOut : 0x00000000
- Check : 0x340BC6D9 («123456789»)

Формат функции:

tkCrc32(crc, buf[], len)

Параметры:

crc	Начальное значение CRC.
buf	Массив байтов, для которого вычисляется CRC.
len	Количество байтов в массиве buf .

Возвращаемое значение:

CRC-32 значение.

Пример вызова функции:

```
new data{9} = "123456789";  
new crc_init = 0xFFFFFFFF;  
new crc = tkCrc32(crc_init, data, 9);  
// Ожидаемый результат: 0x340BC6D9
```

tkCrc8_MinTrans

Вычисление CRC-8.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

- Poly : $0x31 \ x^8 + x^5 + x^4 + 1$
- Init : 0xFF
- Revert : false
- XorOut : 0x00
- Check : 0xF7 («123456789»)

Формат функции:

tkCrc8_MinTrans(buf[], len)

Параметры:

buf	Массив байтов, для которого вычисляется CRC.
len	Количество байтов в массиве buf .

Возвращаемое значение:

CRC-8 значение.

Пример вызова функции:

```
new data{9} = "123456789";  
new crc = tkCrc8_MinTrans(data, 9);  
// Ожидаемый результат: 0xF7
```

tkCrc16_MinTrans

Вычисление CRC-16/CCITT.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

- Poly : $0x1021 \cdot x^{16} + x^{12} + x^5 + 1$
- Init : 0xFFFF
- Revert : false
- XorOut : 0x0000
- Check : 0x29B1 («123456789»)

Формат функции:

tkCrc16_MinTrans(crc, buf[], len)

Параметры:

crc	Начальное значение CRC.
buf	Массив байтов, для которого вычисляется CRC.
len	Количество байтов в массиве buf .

Возвращаемое значение:

CRC-16 значение.

Пример вызова функции:

```
new data{9} = "123456789";  
new crc_init = 0xFFFF;  
new crc = tkCrc16_MinTrans(crc_init, data, 9);  
// Ожидаемый результат: 0x29B1
```

memset

Заполнение указанной области памяти заданным значением.

Для использования функции подключите файл:

```
#include <agMisc.inc>
```

Формат функции:

```
memset(dest[], index=0, numbytes, value=0, maxlenh=sizeof dest)
```

Параметры:

dest[]	Массив, который требуется заполнить.
index	Номер байта в массиве dest[] , с которого требуется начать заполнение (по умолчанию 0). Необязательный параметр.
numbytes	Количество байтов для заполнения.
value	Значение для заполнения (0–255, по умолчанию 0). Необязательный параметр.
maxlength	Максимальный размер массива dest[] . Необязательный параметр.

Возвращаемое значение:

Количество заполненных байтов (целое число).

Пример вызова функции:

```
// Обнуление массива
new buffer[10];
memset(buffer, 0, sizeof(buffer)*4); // Умножаем на 4 (размер ячейки)

// Заполнение со смещением
new data[20];
memset(data, 5*4, 10, 0xFF); // Заполняет 10 байт значением 0xFF начиная с 5-й ячейки

// Частичное заполнение с контролем границ
memset(data, 0, 15, 0xAA, sizeof(data));
```

Функции для работы с навигационными приемниками

Список функций	Описание
tkGetPosition	Получение текущих навигационных данных.
tkSavePoint	Принудительное сохранение точки с координатами.

tkGetPosition

Получение текущих навигационных данных.

Для использования функции подключите файл:

```
#include <agNavigation.inc>
```

Формат функции:

tkGetPosition(position[])

Параметры:

position	Массив структур, куда записываются навигационные данные: <ul style="list-style-type: none">• .lat — широта в градусах;• .lon — долгота в градусах;• .height — высота над уровнем моря, в метрах;• .course — курс (направление) в градусах;• .speed — скорость в км/ч;• .hdop — коэффициент точности HDOP (горизонтальная погрешность);• .numSV — количество спутников.
-----------------	--

Возвращаемое значение:

true — навигационные данные достоверны, false — данные недостоверны.

Пример вызова функции:

```
new pos[Float: .lat, Float: .lon, Float: .height, Float: .course, Float: .speed, Float: .hdop, .numSV];
if (tkGetPosition(pos)) {
    printf("Lat: %f, Lon: %f, H: %f m, C: %f°, V: %f km/h, HDOP: %f, numSV: %d\r\n",
        pos.lat, pos.lon, pos.height, pos.course, pos.speed, pos.hdop, pos.numSV);
}
else {
    printf("Navigation data is unavailable\r\n");
}
```

tkSavePoint

Принудительное сохранение точки с координатами.

Для использования функции подключите файл:

```
#include <agNavigation.inc>
```

Формат функции:

tkSavePoint(secondsToWrite)

Параметры:

secondsToWrite	Количество предыдущих секунд, за которые можно записать координаты. По умолчанию 1 — текущая точка (максимум 5).
-----------------------	---

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkSavePoint(1);
```

Функции для работы с параметрами устройств АвтоГРАФ

Список функций	Описание
<u>tkGetLevelValue</u>	Запрос значения уровня.
<u>tkSetLevelValue</u>	Установка значения уровня.
<u>tkLevelValueUpdated</u>	Проверка актуальности уровневого параметра.
<u>tkGetDiscrValue</u>	Запрос дискретного параметра.
<u>tkSetDiscrValue</u>	Установка дискретного параметра.
<u>tkDiscrValueUpdated</u>	Проверка актуальности дискретного параметра.
<u>tkGetLongValue</u>	Запрос значения длинного параметра.
<u>tkSetLongValue</u>	Установка значения длинного параметра.
<u>tkLongValueUpdated</u>	Проверка актуальности длинного параметра.
<u>tkGetGenericFloatValue</u>	Запрос значения произвольного параметра как числа с плавающей запятой.
<u>tkGetGenericIntValue</u>	Запрос значения произвольного параметра как целочисленного.
<u>tkSetGenericFloatValue</u>	Установка значения произвольного параметра как числа с плавающей запятой.
<u>tkSetGenericIntValue</u>	Установка значения произвольного параметра как целочисленного.
<u>tkGenericValueUpdated</u>	Проверка актуальности произвольного параметра.
<u>tkWriteValue</u>	Запись пользовательских данных в бинарный файл устройства.
<u>tkWriteFloatValue</u>	Запись пользовательских данных типа float в бинарный файл устройства.
<u>tkWriteLongRecord</u>	Запись пользовательского буфера в бинарный файл устройства.
<u>tkSetDeviceFlag</u>	Установка состояния флага устройства.
<u>tkGetDeviceFlag</u>	Чтение текущего состояния битов устройства.
<u>tkWriteTkiaLongRecord</u>	Сохранение пользовательского буфера в запись ТКИА бинарного файла устройства.

Список групп параметров	Описание
<u>DeviceFlags</u>	Флаги (биты состояний) контроллера.
<u>LongParamId</u>	Длинные параметры.
<u>LevelId</u>	Уровневые параметры.
<u>DiscrParamId</u>	Дискретные параметры.
<u>GenericParamsId</u>	Произвольные параметры
<u>ServerConnectionStatus</u>	Статус подключения к серверу.
<u>DangerZoneType</u>	Типы зон в системе приближения.

tkGetLevelValue

Запрос значения уровня.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkGetLevelValue(id)

Параметры:

id	Номер параметра из LevelId.
----	-----------------------------

Возвращаемое значение:

Значение уровня типа Float.

Примечание. Параметры LEVEL_RAM_PARAM_1...LEVEL_RAM_PARAM_10 можно использовать для передачи данных скрипту, меняя их с помощью текстовой команды SETLEVELVALUEn, отправленной через сервер или посредством SMS, AGL и событий.

Пример вызова функции:

```
new Float: value = tkGetLevelValue(VEHICLE_SPEED_SPN_84)
```

tkSetLevelValue

Установка значения уровня.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkSetLevelValue(id, value)

Параметры:

id	Номер параметра из <code>LevelId</code> .
value	Устанавливаемое значение уровня (Float).

Возвращаемое значение:

Нет.

Примечание. Параметр, записанный командой `tkSetLevelValue`, попадает в записи устройства с периодом и логикой того модуля, к которому этот параметр относится. Так, уровни, относящиеся к CAN, будут записаны с периодом записи CAN параметров, уровни, относящиеся к датчикам уровня топлива, будут записаны с периодом записи данных с ДУТ и так далее.

Некоторые уровневые параметры, например, `LEVEL_RAM_PARAM_n`, в записи устройства не попадают, но, как и другие параметры, могут использоваться для настройки адаптивов, в событиях и в Т.Скрипт.

Если требуется гарантированно записать определенное значение в определенный момент, то следует использовать функции `tkWriteValue` и `tkWriteFloatValue`.

Примечание. Некоторые параметры, например, напряжение аналогового входа, автоматически перезаписываются прошивкой устройства, и запись в такие параметры функцией `tkSetLevelValue` не имеет смысла.

Примечание. Параметры `LEVEL_RAM_PARAM_1...LEVEL_RAM_PARAM_10` можно использовать для передачи данных скрипту, меняя их с помощью текстовой команды `SETLEVELVALUEn`, отправленной через сервер или посредством SMS, AGL и событий.

Пример вызова функции:

```
tkSetLevelValue(VEHICLE_SPEED_SPN_84, 2.75)
```

tkLevelValueUpdated

Проверка актуальности уровневого параметра.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkLevelValueUpdated(id, bool: reset)

Параметры:

id	Номер параметра из <u>LevelId</u> .
reset	Флаг сброса актуальности после выполнения функции.

Возвращаемое значение:

true — значение параметра было обновлено после последнего сброса, false — нет.

Пример вызова функции:

```
tkLevelValueUpdated(VEHICLE_SPEED_SPN_84, true);
```

tkGetDiscrValue

Запрос дискретного параметра.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkGetDiscrValue(id)

Параметры:

id	Номер параметра из DiscrParamId.
----	----------------------------------

Возвращаемое значение:

Значение параметра в uint32_t.

Пример вызова функции:

```
new DiscrValue = tkGetDiscrValue(DISCR_RAM_PARAM_1)
```

tkSetDiscrValue

Установка дискретного параметра.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkSetDiscrValue(id, value)

Параметры:

id	Номер параметра из DiscrParamId.
value	Устанавливаемое значение параметра в uint32_t.

Возвращаемое значение:

Нет.

Примечание. Параметр, записанный командой `tkSetDiscrValue`, попадает в запись устройства с периодом и логикой того модуля, к которому этот параметр относится. Значительная часть дискретных параметров относится к шине CAN и будет записана с периодом записи CAN параметров.

Некоторые дискретные параметры, например, `DISCR_RAM_PARAM_n`, в записи устройства не попадают, но, как и другие параметры, могут использоваться для настройки адаптивов, в событиях и в Т.Скрипт.

Если требуется гарантированно записать определенное значение в определенный момент, то следует использовать функцию `tkWriteValue`.

Примечание. Некоторые параметры, например, состояние выхода, являются справочными и автоматически перезаписываются прошивкой устройства, и запись в такие параметры функцией `tkSetDiscrValue` не имеет смысла.

Примечание. Параметры `DISCR_RAM_PARAM_1...DISCR_RAM_PARAM_10` можно использовать для передачи данных скрипту, меняя их с помощью текстовой команды `SETDISCRVALUEn`, отправленной через сервер или посредством SMS, AGL и событий.

Пример вызова функции:

```
tkSetDiscrValue(DISCR_RAM_PARAM_1, 42)
```

tkDiscrValueUpdated

Проверка актуальности дискретного параметра.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkDiscrValueUpdated(id, bool: reset)

Параметры:

id	Номер параметра из DiscrParamId.
reset	Флаг сброса актуальности после выполнения функции.

Возвращаемое значение:

true — значение параметра было обновлено после последнего сброса, false — нет.

Примечание. Параметры DISCR_RAM_PARAM_1...DISCR_RAM_PARAM_10 можно использовать для передачи данных скрипту, меняя их с помощью текстовой команды SETDISCRVALUEn, отправленной через сервер или посредством SMS, AGL и событий.

Пример вызова функции:

```
tkDiscrValueUpdated(DISCR_RAM_PARAM_1, true);
```

tkGetLongValue

Запрос значения длинного параметра.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkGetLongValue(id)

Параметры:

id	Номер параметра из LongParamId.
----	---------------------------------

Возвращаемое значение:

Значение типа Float.

Пример вызова функции:

```
new Float: value = tkGetLongValue(TOTAL_VEHICLE_DISTANCE_SPN_917)
```

tkSetLongValue

Установка значения длинного параметра.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkSetLongValue(id, value)

Параметры:

id	Номер параметра из LongParamId.
value	Устанавливаемое значение длинного параметра (Float).

Возвращаемое значение:

Нет.

Примечание. Параметр, записанный командой `tkSetLongValue`, попадает в записи устройства с периодом записи CAN параметров.

Если требуется гарантированно записать определенное значение в определенный момент, то следует использовать функции `tkWriteValue` и `tkWriteFloatValue`.

Пример вызова функции:

```
tkSetLongValue(TOTAL_VEHICLE_DISTANCE_SPN_917, 2.75)
```

tkLongValueUpdated

Проверка актуальности длинного параметра.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkLongValueUpdated(id, bool: reset)

Параметры:

id	Номер параметра из LongParamId.
reset	Флаг сброса актуальности после выполнения функции.

Возвращаемое значение:

true — значение параметра было обновлено после последнего сброса, false — нет.

Пример вызова функции:

```
tkLongValueUpdated(TOTAL_VEHICLE_DISTANCE_SPN_917, true);
```

tkGetGenericFloatValue

Запрос значения произвольного параметра как числа с плавающей запятой.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkGetGenericFloatValue(id)

Параметры:

id	Номер параметра из GenericParamsId.
----	-------------------------------------

Возвращаемое значение:

Значение типа Float.

Пример вызова функции:

```
new Float: value = tkGetGenericFloatValue(GENERIC_CAN_PARAM_1)
```

tkGetGenericIntValue

Запрос значения произвольного параметра как целочисленного.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkGetGenericIntValue(id)

Параметры:

id	Номер параметра из GenericParamsId.
----	-------------------------------------

Возвращаемое значение:

Значение типа int.

Пример вызова функции:

```
new value = tkGetGenericIntValue(GENERIC_CAN_PARAM_6)
```

tkSetGenericFloatValue

Установка значения произвольного параметра как числа с плавающей запятой.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkSetGenericFloatValue(id, value)

Параметры:

id	Номер параметра из <code>GenericParamsId</code> .
value	Устанавливаемое значение произвольного параметра (Float).

Возвращаемое значение:

Нет.

Примечание. Параметр, записанный командой `tkSetGenericFloatValue` с **id** в диапазоне от `GENERIC_CAN_PARAM_1` до `GENERIC_CAN_PARAM_50`, попадает в записи устройства с периодом записи CAN параметров. Параметр, записанный командой `tkSetGenericFloatValue` с **id** в диапазоне от `GENERIC_MODBUS_PARAM_1` до `GENERIC_MODBUS_PARAM_100`, попадает в записи устройства с периодом записи параметров MODBUS.

При записи произвольный параметр будет преобразован с учетом настроек и коэффициентов, заданных в устройстве для соответствующего параметра, поэтому записанное значение может отличаться от установленного.

Если требуется гарантированно записать определенное значение в определенный момент, то следует использовать функции `tkWriteValue` и `tkWriteFloatValue`.

Пример вызова функции:

```
tkSetGenericFloatValue(GENERIC_CAN_PARAM_6, 1.5)
```

tkSetGenericIntValue

Установка значения произвольного параметра как целочисленного.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkSetGenericIntValue(id, value)

Параметры:

id	Номер параметра из <code>GenericParamsId</code> .
value	Устанавливаемое значение произвольного параметра.

Примечание. Параметр, записанный командой `tkSetGenericFloatValue` с **id** в диапазоне от `GENERIC_CAN_PARAM_1` до `GENERIC_CAN_PARAM_50`, попадает в записи устройства с периодом записи CAN параметров. Параметр, записанный командой `tkSetGenericFloatValue` с **id** в диапазоне от `GENERIC_MODBUS_PARAM_1` до `GENERIC_MODBUS_PARAM_100`, попадает в записи устройства с периодом записи параметров MODBUS.

Если требуется гарантированно записать определенное значение в определенный момент, то следует использовать функции `tkWriteValue` и `tkWriteFloatValue`.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkSetGenericIntValue(GENERIC_CAN_PARAM_6, 0xface)
```

tkGenericValueUpdated

Проверка актуальности произвольного параметра.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkGenericValueUpdated(id, bool: reset)

Параметры:

id	Номер параметра из <u>GenericParamsId</u> .
reset	Флаг сброса актуальности после выполнения функции.

Возвращаемое значение:

true — значение параметра было обновлено после последнего сброса, false — нет.

Пример вызова функции:

```
tkGenericValueUpdated(GENERIC_CAN_PARAM_1, true);
```

tkWriteValue

Запись пользовательских данных в бинарный файл устройства.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Делает запись пользовательского значения в бинарный файл устройства.

Формат функции:

tkWriteValue(paramId, paramValue)

Параметры:

paramId	ID параметра, должен быть в диапазоне 0...65535 (0xffff).
paramValue	Значение параметра, 0...4294967295 (0xFFFFFFFF).

Возвращаемое значение:

Нет.

Примечание. Значение параметра **paramValue** будет записано в момент вызова функции в прочий числовой параметр с индексом ($0xFA0000 + \text{paramId}$).

Примечание. Функции `tkWriteValue` и `tkWriteFloatValue` записывают прочие числовые параметры с индексами в одинаковом диапазоне. Разделение по типу идет в ПО верхнего уровня.

Примечание. В диспетчерском ПО АвтоГРАФ.Pro значение параметра можно получить с помощью вызова функции `NDUInt(0xFA0000 + paramId)`

Пример вызова функции:

```
tkWriteValue(10, 556677)
```

tkWriteFloatValue

Запись пользовательских данных типа float в бинарный файл устройства.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Делает запись пользовательского значения в бинарный файл устройства.

Формат функции:

tkWriteFloatValue(paramId, Float:paramValue)

Параметры:

paramId	ID параметра, должен быть в диапазоне 0...65535 (0xffff).
paramValue	Значение параметра (Float).

Возвращаемое значение:

Нет.

Примечание. Значение параметра **paramValue** будет записано в момент вызова функции в прочий числовой параметр с индексом ($0xFA0000 + \mathbf{paramId}$) в формате IEEE 754 одинарной точности (32 бит).

Примечание. Функции tkWriteValue и tkWriteFloatValue записывают прочие числовые параметры с индексами в одинаковом диапазоне. Разделение по типу идет в ПО верхнего уровня.

Примечание. В диспетчерском ПО АвтоГРАФ.Pro значение параметра можно получить с помощью вызова функции NDFloat($0xFA0000 + \mathbf{paramId}$)

Пример вызова функции:

```
tkWriteFloatValue(11, 1.5)
```

tkWriteLongRecord

Запись пользовательского буфера в бинарный файл устройства.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Делает запись пользовательского бинарного массива или строки в бинарный файл устройства.

Формат функции:

tkWriteLongRecord (paramId, buf{}, bufSize, bool:pack=true)

Параметры:

paramId	ID параметра, должен быть в диапазоне 0...65535 (0xffff).
buf	Записываемые данные.
bufSize	Длина (размер) записываемых данных, 1...1530 байт.
pack	Флаг упаковки приемного буфера.

Возвращаемое значение:

Нет.

Примечание. Содержимое буфера **buf** будет записано в момент вызова функции в длинную запись с индексом (0xFA0000 + **paramId**).

Пример вызова функции:

```
tkWriteLongRecord(10, "Script started", 14, true)
```

tkSetDeviceFlag

Установка состояния флага устройства.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkSetDeviceFlag(flags, status)

Параметры:

flags	Битовая маска (см. DeviceFlags).
status	true — установить флаг, false — сбросить флаг.

Возвращаемое значение:

Нет.

Примечание. Флаги устройства записываются в каждую запись устройства, и могут быть использованы в настройках событий.

Примечание. Некоторые флаги, например, наличие основного питания, автоматически перезаписывается прошивкой устройства, и запись в такие параметры функцией `tkSetDeviceFlag` не имеет смысла.

Пример вызова функции:

```
tkSetDeviceFlag(DF_MASK_ALARM | DF_MASK_GSM, true);
```

tkGetDeviceFlag

Чтение текущего состояния битов устройства.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Формат функции:

tkGetDeviceFlag(flags)

Параметры:

flags	Битовая маска (см. DeviceFlags).
--------------	----------------------------------

Возвращаемое значение:

Набор установленных битов из параметра **flags**.

Пример вызова функции:

```
new state = tkGetDeviceFlag(DF_MASK_BORT | DF_MASK_USBCONNECT);  
if (state & DF_MASK_BORT) {}
```

tkWriteTkiaLongRecord

Сохранение пользовательского буфера в запись ТКІА бинарного файла устройства.

Для использования функции подключите файл:

```
#include <agParams.inc>
```

Сохраняет данные из пользовательского бинарного массива или строки в запись ТКІА бинарного файла устройства.

Формат функции:

tkWriteTkiaLongRecord (buf{}, bufSize, bool:pack=true))

Параметры:

buf	Записываемые данные.
bufSize	Длина (размер) записываемых данных, 1...1530 байт.
pack	Флаг упаковки приемного буфера.

Возвращаемое значение:

Нет.

Примечание. Содержимое буфера **buf** будет сохранено в длинную запись с индексом 0x50 в момент вызова функции.

Пример вызова функции:

```
tkWriteTkiaLongRecord("Script started", 14, true)
```

DeviceFlags

Флаги (биты состояний) контроллера.

DF_MASK_READ1 = 0x00000100	9 — данные отправлены на первый сервер (только в записях).
DF_MASK_READ2 = 0x00000200	10 — данные отправлены на второй сервер (только в записях).
DF_MASK_BORT = 0x00000400	11 — наличие основного питания.
DF_MASK_RESERV = 0x00000800	12 — наличие питания от внешнего аккумулятора.
DF_MASK_INANTOK = 0x00001000	13 — состояние антенны навигационного приемника.
DF_MASK_OUTANTOK = 0x00002000	14 — зарезервировано.
DF_MASK_USBCONNECT = 0x00004000	15 — подключено USB.
DF_MASK_ALARM = 0x00008000	16 — нажата тревожная кнопка.
DF_MASK_RPMCAN = 0x00010000	17 — есть обороты по CAN.
DF_MASK_ROAMING = 0x00020000	18 — контроллер находится в роуминге.
DF_MASK_LOADING = 0x00040000	19 — идет погрузка в транспортное средство.
DF_MASK_GSM = 0x00080000	20 — наличие GSM сигнала.
DF_MASK_ISSTAND = 0x00100000	21 — остановка.
DF_MASK_AKK_IN = 0x00400000	23 — наличие питания от внутреннего аккумулятора.
DF_MASK_READ3 = 0x00800000	24 — данные отправлены на третий сервер (только в записях).

LongParamId

Длинные параметры.

LONG_INVALID_PARAM = 0	0 — не используется.
TOTAL_FUEL_USED_SPN_250	1 — суммарно использованное топливо, л.
SERVICE_DISTANCE_SPN_914	2 — пробег до ТО, км.
ENGINE_HOURS_SPN_247	3 — моточасы, ч.
TOTAL_VEHICLE_DISTANCE_SPN_917	4 — полный пробег, м.
TRIP_DISTANCE_SPN_918	5 — пробег за поездку, м.
CALCULATED_FUEL_CONSUMPTION	6 — потребление топлива, вычисленное по мгновенному расходу с прошлой записи, л.

LevelId

Уровневые параметры.

LEVEL_INVALID_PARAM = 0	0 — не используется.
LEVEL_LLS1	1 — уровень топлива с датчика 1, единицы измерения датчика.
LEVEL_LLS2	2 — уровень топлива с датчика 2, единицы измерения датчика.
LEVEL_LLS3	3 — уровень топлива с датчика 3, единицы измерения датчика.
LEVEL_LLS4	4 — уровень топлива с датчика 4, единицы измерения датчика.
LEVEL_LLS5	5 — уровень топлива с датчика 5, единицы измерения датчика.
LEVEL_LLS6	6 — уровень топлива с датчика 6, единицы измерения датчика.
LEVEL_LLS7	7 — уровень топлива с датчика 7, единицы измерения датчика.
LEVEL_LLS8	8 — уровень топлива с датчика 8, единицы измерения датчика.
LEVEL_TEMP1	9 — температура с датчика 1, °C.
LEVEL_TEMP2	10 — температура с датчика 2, °C.
LEVEL_TEMP3	11 — температура с датчика 3, °C.
LEVEL_TEMP4	12 — температура с датчика 4, °C.
LEVEL_TEMP5	13 — температура с датчика 5, °C.
LEVEL_TEMP6	14 — температура с датчика 6, °C.
LEVEL_TEMP7	15 — температура с датчика 7, °C.
LEVEL_TEMP8	16 — температура с датчика 8, °C.
LEVEL_INT_CPU_TEMPSENSOR	17 — температура МК, °C.
LEVEL_VREFINT	18 — напряжение внутренней опоры, В.
LEVEL_GNS_ANT_VDD	19 — напряжение антенны, В. Статический адаптив.
LEVEL_EXT_VDD	20 — напряжение внешнего питания, В.
LEVEL_A_IN_1	21 — напряжение аналогового входа 1, В.
LEVEL_A_IN_2	22 — напряжение аналогового входа 2, В.
LEVEL_A_EXT_BATTERY	23 — напряжение внешнего аккумулятора, В.
LEVEL_A_INT_BATTERY	24 — напряжение внутреннего аккумулятора, В.
VEHICLE_SPEED_SPN_84	25 — скорость, км/ч.
ACCEL_PEDAL_SPN_91	26 — педаль акселератора, %.
FUEL_LEVEL_1_SPN_96	27 — уровень топлива 1, %.
FUEL_LEVEL_2_SPN_96	28 — уровень топлива 2, %.
FUEL_LEVEL_3_SPN_96	29 — уровень топлива 3, %.
FUEL_LEVEL_4_SPN_96	30 — уровень топлива 4, %.
FUEL_LEVEL_5_SPN_96	31 — уровень топлива 5, %.
FUEL_LEVEL_6_SPN_96	32 — уровень топлива 6, %.

ADBLUE_LEVEL_SPN_1761	33 — уровень AdBlue, %.
RPM_SPN_190	34 — обороты, об/бит.
OIL_PRESSURE_SPN_100	35 — давление масла, кПа.
OIL_TEMP_SPN_175	36 — температура масла, °С.
COOLANT_TEMP_SPN_110	37 — температура охлаждающей жидкости, °С.
FUEL_TEMP_SPN_174	38 — температура топлива, °С.
AMBIENT_AIR_TEMP_SPN_171	39 — внешняя температура, °С.
CHARGER_AIR_TEMP_SPN_105	40 — температура в коллекторе наддува, °С.
ENGINE_AIR_INLET_PRES_SPN_106	41 — давление воздуха на впуске, кПа.
ENGINE_CHGR_BOOST_PRES_SPN_102	42 — избыточное давление наддува, кПа.
LEVEL_WHEEL_LOAD_1_1	43 — нагрузка на колесо 1 оси 1, кг.
LEVEL_WHEEL_LOAD_1_2	44 — нагрузка на колесо 2 оси 1, кг.
LEVEL_WHEEL_LOAD_1_3	45 — нагрузка на колесо 3 оси 1, кг.
LEVEL_WHEEL_LOAD_1_4	46 — нагрузка на колесо 4 оси 1, кг.
LEVEL_WHEEL_LOAD_1_5	47 — нагрузка на колесо 5 оси 1, кг.
LEVEL_WHEEL_LOAD_1_6	48 — нагрузка на колесо 6 оси 1, кг.
LEVEL_WHEEL_LOAD_2_1	49 — нагрузка на колесо 1 оси 2, кг.
LEVEL_WHEEL_LOAD_2_2	50 — нагрузка на колесо 2 оси 2, кг.
LEVEL_WHEEL_LOAD_2_3	51 — нагрузка на колесо 3 оси 2, кг.
LEVEL_WHEEL_LOAD_2_4	52 — нагрузка на колесо 4 оси 2, кг.
LEVEL_WHEEL_LOAD_2_5	53 — нагрузка на колесо 5 оси 2, кг.
LEVEL_WHEEL_LOAD_2_6	54 — нагрузка на колесо 6 оси 2, кг.
LEVEL_WHEEL_LOAD_3_1	55 — нагрузка на колесо 1 оси 3, кг.
LEVEL_WHEEL_LOAD_3_2	56 — нагрузка на колесо 2 оси 3, кг.
LEVEL_WHEEL_LOAD_3_3	57 — нагрузка на колесо 3 оси 3, кг.
LEVEL_WHEEL_LOAD_3_4	58 — нагрузка на колесо 4 оси 3, кг.
LEVEL_WHEEL_LOAD_3_5	59 — нагрузка на колесо 5 оси 3, кг.
LEVEL_WHEEL_LOAD_3_6	60 — нагрузка на колесо 6 оси 3, кг.
LEVEL_WHEEL_LOAD_4_1	61 — нагрузка на колесо 1 оси 4, кг.
LEVEL_WHEEL_LOAD_4_2	62 — нагрузка на колесо 2 оси 4, кг.
LEVEL_WHEEL_LOAD_4_3	63 — нагрузка на колесо 3 оси 4, кг.
LEVEL_WHEEL_LOAD_4_4	64 — нагрузка на колесо 4 оси 4, кг.
LEVEL_WHEEL_LOAD_4_5	65 — нагрузка на колесо 5 оси 4, кг.
LEVEL_WHEEL_LOAD_4_6	66 — нагрузка на колесо 6 оси 4, кг.
LEVEL_WHEEL_LOAD_5_1	67 — нагрузка на колесо 1 оси 5, кг.
LEVEL_WHEEL_LOAD_5_2	68 — нагрузка на колесо 2 оси 5, кг.

LEVEL_WHEEL_LOAD_5_3	69 — нагрузка на колесо 3 оси 5, кг.
LEVEL_WHEEL_LOAD_5_4	70 — нагрузка на колесо 4 оси 5, кг.
LEVEL_WHEEL_LOAD_5_5	71 — нагрузка на колесо 5 оси 5, кг.
LEVEL_WHEEL_LOAD_5_6	72 — нагрузка на колесо 6 оси 5, кг.
LEVEL_WHEEL_LOAD_6_1	73 — нагрузка на колесо 1 оси 6, кг.
LEVEL_WHEEL_LOAD_6_2	74 — нагрузка на колесо 2 оси 6, кг.
LEVEL_WHEEL_LOAD_6_3	75 — нагрузка на колесо 3 оси 6, кг.
LEVEL_WHEEL_LOAD_6_4	76 — нагрузка на колесо 4 оси 6, кг.
LEVEL_WHEEL_LOAD_6_5	77 — нагрузка на колесо 5 оси 6, кг.
LEVEL_WHEEL_LOAD_6_6	78 — нагрузка на колесо 6 оси 6, кг.
LEVEL_WHEEL_LOAD_7_1	79 — нагрузка на колесо 1 оси 7, кг.
LEVEL_WHEEL_LOAD_7_2	80 — нагрузка на колесо 2 оси 7, кг.
LEVEL_WHEEL_LOAD_7_3	81 — нагрузка на колесо 3 оси 7, кг.
LEVEL_WHEEL_LOAD_7_4	82 — нагрузка на колесо 4 оси 7, кг.
LEVEL_WHEEL_LOAD_7_5	83 — нагрузка на колесо 5 оси 7, кг.
LEVEL_WHEEL_LOAD_7_6	84 — нагрузка на колесо 6 оси 7, кг.
LEVEL_WHEEL_LOAD_8_1	85 — нагрузка на колесо 1 оси 8, кг.
LEVEL_WHEEL_LOAD_8_2	86 — нагрузка на колесо 2 оси 8, кг.
LEVEL_WHEEL_LOAD_8_3	87 — нагрузка на колесо 3 оси 8, кг.
LEVEL_WHEEL_LOAD_8_4	88 — нагрузка на колесо 4 оси 8, кг.
LEVEL_WHEEL_LOAD_8_5	89 — нагрузка на колесо 5 оси 8, кг.
LEVEL_WHEEL_LOAD_8_6	90 — нагрузка на колесо 6 оси 8, кг.
LEVEL_WHEEL_LOAD_9_1	91 — нагрузка на колесо 1 оси 9, кг.
LEVEL_WHEEL_LOAD_9_2	92 — нагрузка на колесо 2 оси 9, кг.
LEVEL_WHEEL_LOAD_9_3	93 — нагрузка на колесо 3 оси 9, кг.
LEVEL_WHEEL_LOAD_9_4	94 — нагрузка на колесо 4 оси 9, кг.
LEVEL_WHEEL_LOAD_9_5	95 — нагрузка на колесо 5 оси 9, кг.
LEVEL_WHEEL_LOAD_9_6	96 — нагрузка на колесо 6 оси 9, кг.
LEVEL_WHEEL_LOAD_10_1	97 — нагрузка на колесо 1 оси 10, кг.
LEVEL_WHEEL_LOAD_10_2	98 — нагрузка на колесо 2 оси 10, кг.
LEVEL_WHEEL_LOAD_10_3	99 — нагрузка на колесо 3 оси 10, кг.
LEVEL_WHEEL_LOAD_10_4	100 — нагрузка на колесо 4 оси 10, кг.
LEVEL_WHEEL_LOAD_10_5	101 — нагрузка на колесо 5 оси 10, кг.
LEVEL_WHEEL_LOAD_10_6	102 — нагрузка на колесо 6 оси 10, кг.
LEVEL_WHEEL_LOAD_11_1	103 — нагрузка на колесо 1 оси 11, кг.
LEVEL_WHEEL_LOAD_11_2	104 — нагрузка на колесо 2 оси 11, кг.

LEVEL_WHEEL_LOAD_11_3	105 — нагрузка на колесо 3 оси 11, кг.
LEVEL_WHEEL_LOAD_11_4	106 — нагрузка на колесо 4 оси 11, кг.
LEVEL_WHEEL_LOAD_11_5	107 — нагрузка на колесо 5 оси 11, кг.
LEVEL_WHEEL_LOAD_11_6	108 — нагрузка на колесо 6 оси 11, кг.
LEVEL_WHEEL_LOAD_12_1	109 — нагрузка на колесо 1 оси 12, кг.
LEVEL_WHEEL_LOAD_12_2	110 — нагрузка на колесо 2 оси 12, кг.
LEVEL_WHEEL_LOAD_12_3	111 — нагрузка на колесо 3 оси 12, кг.
LEVEL_WHEEL_LOAD_12_4	112 — нагрузка на колесо 4 оси 12, кг.
LEVEL_WHEEL_LOAD_12_5	113 — нагрузка на колесо 5 оси 12, кг.
LEVEL_WHEEL_LOAD_12_6	114 — нагрузка на колесо 6 оси 12, кг.
LEVEL_WHEEL_LOAD_13_1	115 — нагрузка на колесо 1 оси 13, кг.
LEVEL_WHEEL_LOAD_13_2	116 — нагрузка на колесо 2 оси 13, кг.
LEVEL_WHEEL_LOAD_13_3	117 — нагрузка на колесо 3 оси 13, кг.
LEVEL_WHEEL_LOAD_13_4	118 — нагрузка на колесо 4 оси 13, кг.
LEVEL_WHEEL_LOAD_13_5	119 — нагрузка на колесо 5 оси 13, кг.
LEVEL_WHEEL_LOAD_13_6	120 — нагрузка на колесо 6 оси 13, кг.
LEVEL_WHEEL_LOAD_14_1	121 — нагрузка на колесо 1 оси 14, кг.
LEVEL_WHEEL_LOAD_14_2	122 — нагрузка на колесо 2 оси 14, кг.
LEVEL_WHEEL_LOAD_14_3	123 — нагрузка на колесо 3 оси 14, кг.
LEVEL_WHEEL_LOAD_14_4	124 — нагрузка на колесо 4 оси 14, кг.
LEVEL_WHEEL_LOAD_14_5	125 — нагрузка на колесо 5 оси 14, кг.
LEVEL_WHEEL_LOAD_14_6	126 — нагрузка на колесо 6 оси 14, кг.
LEVEL_WHEEL_LOAD_15_1	127 — нагрузка на колесо 1 оси 15, кг.
LEVEL_WHEEL_LOAD_15_2	128 — нагрузка на колесо 2 оси 15, кг.
LEVEL_WHEEL_LOAD_15_3	129 — нагрузка на колесо 3 оси 15, кг.
LEVEL_WHEEL_LOAD_15_4	130 — нагрузка на колесо 4 оси 15, кг.
LEVEL_WHEEL_LOAD_15_5	131 — нагрузка на колесо 5 оси 15, кг.
LEVEL_WHEEL_LOAD_15_6	132 — нагрузка на колесо 6 оси 15, кг.
LEVEL_WHEEL_LOAD_16_1	133 — нагрузка на колесо 1 оси 16, кг.
LEVEL_WHEEL_LOAD_16_2	134 — нагрузка на колесо 2 оси 16, кг.
LEVEL_WHEEL_LOAD_16_3	135 — нагрузка на колесо 3 оси 16, кг.
LEVEL_WHEEL_LOAD_16_4	136 — нагрузка на колесо 4 оси 16, кг.
LEVEL_WHEEL_LOAD_16_5	137 — нагрузка на колесо 5 оси 16, кг.
LEVEL_WHEEL_LOAD_16_6	138 — нагрузка на колесо 6 оси 16, кг.
ENGINE_FUEL_RATE_SPN_183	139 — расход топлива в единицу времени, л/ч.
ENGINE_THROTTLE_POS_SPN_51	140 — положение дроссельной заслонки, %.

ACTUAL_ENG_PCNT_TORQUE_SPN_513	141 — действующий момент, %.
CRUISE_CONTROL_SET_SPEED_SPN_86	142 — скорость круиз-контроля, км/ч.
NOMINAL_FRICT_PCNT_TRQ_SPN_514	143 — номинальное трение — % крутящего момента, %.
BATTERY_VOLTAGE_SPN_158	144 — напряжение АКБ, В.
BAROMETRIC_PRESSURE_SPN_108	145 — абсолютное атмосферное давление, кПа.
ENGINE_LOAD_SPN_92	146 — нагрузка на двигатель, %.
BATTERY_CURRENT_SPN_114	147 — ток АКБ, А.
PARTICULATE_FILTER	148 — сажевый фильтр, %.
LEVEL_FREQUENCY_01	149 — частота или ШИМ со входа 1.
LEVEL_FREQUENCY_02	150 — частота или ШИМ со входа 2.
LEVEL_FREQUENCY_03	151 — частота или ШИМ со входа 3.
LEVEL_FREQUENCY_04	152 — частота или ШИМ со входа 4.
LEVEL_FREQUENCY_05	153 — частота или ШИМ со входа 5.
LEVEL_FREQUENCY_06	154 — частота или ШИМ со входа 6.
LEVEL_FREQUENCY_07	155 — частота или ШИМ со входа 7.
LEVEL_FREQUENCY_08	156 — частота или ШИМ со входа 8.
LEVEL_FREQUENCY_09	157 — частота или ШИМ со входа 9.
LEVEL_FREQUENCY_RPM	158 — частота или ШИМ со входа RPM.
LEVEL_LLS_1_TEMPERATURE	159 — температура с ДУТ 1, °C, (знаковые, 8 бит).
LEVEL_LLS_2_TEMPERATURE	160 — температура с ДУТ 2, °C.
LEVEL_LLS_3_TEMPERATURE	161 — температура с ДУТ 3, °C.
LEVEL_LLS_4_TEMPERATURE	162 — температура с ДУТ 4, °C.
LEVEL_LLS_5_TEMPERATURE	163 — температура с ДУТ 5, °C.
LEVEL_LLS_6_TEMPERATURE	164 — температура с ДУТ 6, °C.
LEVEL_LLS_7_TEMPERATURE	165 — температура с ДУТ 7, °C.
LEVEL_LLS_8_TEMPERATURE	166 — температура с ДУТ 8, °C.
LEVEL_LLS_1_ANGLE	167 — угол с ДУТ 1, градусы (беззнаковые, 0...180°, 8 бит).
LEVEL_LLS_2_ANGLE	168 — угол с ДУТ 2, градусы.
LEVEL_LLS_3_ANGLE	169 — угол с ДУТ 3, градусы.
LEVEL_LLS_4_ANGLE	170 — угол с ДУТ 4, градусы.
LEVEL_LLS_5_ANGLE	171 — угол с ДУТ 5, градусы.
LEVEL_LLS_6_ANGLE	172 — угол с ДУТ 6, градусы.
LEVEL_LLS_7_ANGLE	173 — угол с ДУТ 7, градусы.
LEVEL_LLS_8_ANGLE	174 — угол с ДУТ 8, градусы.
LEVEL_LLS_1_PITCH	175 — угол тангажа с ДУТ 1, градусы (знаковые, -90°...90°, 8 бит).
LEVEL_LLS_2_PITCH	176 — угол тангажа с ДУТ 2, градусы.

LEVEL_LLS_3_PITCH	177 — угол тангажа с ДУТ 3, градусы.
LEVEL_LLS_4_PITCH	178 — угол тангажа с ДУТ 4, градусы.
LEVEL_LLS_5_PITCH	179 — угол тангажа с ДУТ 5, градусы.
LEVEL_LLS_6_PITCH	180 — угол тангажа с ДУТ 6, градусы.
LEVEL_LLS_7_PITCH	181 — угол тангажа с ДУТ 7, градусы.
LEVEL_LLS_8_PITCH	182 — угол тангажа с ДУТ 8, градусы.
LEVEL_LLS_1_ROLL	183 — угол крена с ДУТ 1, градусы (знаковые, -90° ... 90° , 8 бит).
LEVEL_LLS_2_ROLL	184 — угол крена с ДУТ 2, градусы.
LEVEL_LLS_3_ROLL	185 — угол крена с ДУТ 3, градусы.
LEVEL_LLS_4_ROLL	186 — угол крена с ДУТ 4, градусы.
LEVEL_LLS_5_ROLL	187 — угол крена с ДУТ 5, градусы.
LEVEL_LLS_6_ROLL	188 — угол крена с ДУТ 6, градусы.
LEVEL_LLS_7_ROLL	189 — угол крена с ДУТ 7, градусы.
LEVEL_LLS_8_ROLL	190 — угол крена с ДУТ 8, градусы.
LEVEL_LLS_1_FREQUENCY	191 — частота с ДУТ 1, Гц.
LEVEL_LLS_2_FREQUENCY	192 — частота с ДУТ 2, Гц.
LEVEL_LLS_3_FREQUENCY	193 — частота с ДУТ 3, Гц.
LEVEL_LLS_4_FREQUENCY	194 — частота с ДУТ 4, Гц.
LEVEL_LLS_5_FREQUENCY	195 — частота с ДУТ 5, Гц.
LEVEL_LLS_6_FREQUENCY	196 — частота с ДУТ 6, Гц.
LEVEL_LLS_7_FREQUENCY	197 — частота с ДУТ 7, Гц.
LEVEL_LLS_8_FREQUENCY	198 — частота с ДУТ 8, Гц.
LEVEL_A_IN_3	199 — напряжение аналогового входа 3, В.
LEVEL_A_IN_4	200 — напряжение аналогового входа 4, В.
LEVEL_COUPLER_LOAD	201 — нагрузка на сцепное устройство, кг.
LEVEL_CARGO_WEIGHT_SPN_181	202 — вес груза, кг.
LEVEL_TRAILER_WEIGHT_SPN_180	203 — вес трейлера (прицепа), кг.
LEVEL_A_IN_5	204 — напряжение аналогового входа 5, В.
LEVEL_A_IN_6	205 — напряжение аналогового входа 6, В.
LEVEL_NAV_SPEED	206 — скорость с навигационного приемника, км/ч.
LEVEL_LLS_1_BAT_VOLT	207 — напряжение батареи беспроводного ДУТ 1, В.
LEVEL_LLS_2_BAT_VOLT	208 — напряжение батареи беспроводного ДУТ 2, В.
LEVEL_LLS_3_BAT_VOLT	209 — напряжение батареи беспроводного ДУТ 3, В.
LEVEL_LLS_4_BAT_VOLT	210 — напряжение батареи беспроводного ДУТ 4, В.
LEVEL_LLS_5_BAT_VOLT	211 — напряжение батареи беспроводного ДУТ 5, В.
LEVEL_LLS_6_BAT_VOLT	212 — напряжение батареи беспроводного ДУТ 6, В.

LEVEL_LLS_7_BAT_VOLT	213 — напряжение батареи беспроводного ДУТ 7, В.
LEVEL_LLS_8_BAT_VOLT	214 — напряжение батареи беспроводного ДУТ 8, В.
LEVEL_LLS_1_RSSI	215 — RSSI беспроводного ДУТ 1.
LEVEL_LLS_2_RSSI	216 — RSSI беспроводного ДУТ 2.
LEVEL_LLS_3_RSSI	217 — RSSI беспроводного ДУТ 3.
LEVEL_LLS_4_RSSI	218 — RSSI беспроводного ДУТ 4.
LEVEL_LLS_5_RSSI	219 — RSSI беспроводного ДУТ 5.
LEVEL_LLS_6_RSSI	220 — RSSI беспроводного ДУТ 6.
LEVEL_LLS_7_RSSI	221 — RSSI беспроводного ДУТ 7.
LEVEL_LLS_8_RSSI	222 — RSSI беспроводного ДУТ 8.
TKAM_1_ANGLE	223 — угол с датчика угла наклона (TKAM) 1.
TKAM_2_ANGLE	224 — угол с датчика угла наклона (TKAM) 2.
TKAM_3_ANGLE	225 — угол с датчика угла наклона (TKAM) 3.
TKAM_4_ANGLE	226 — угол с датчика угла наклона (TKAM) 4.
TKAM_5_ANGLE	227 — угол с датчика угла наклона (TKAM) 5.
TKAM_6_ANGLE	228 — угол с датчика угла наклона (TKAM) 6.
TKAM_7_ANGLE	229 — угол с датчика угла наклона (TKAM) 7.
TKAM_8_ANGLE	230 — угол с датчика угла наклона (TKAM) 8.
TKAM_9_ANGLE	231 — угол с датчика угла наклона (TKAM) 9.
TKAM_10_ANGLE	232 — угол с датчика угла наклона (TKAM) 10.
TKAM_11_ANGLE	233 — угол с датчика угла наклона (TKAM) 11.
TKAM_12_ANGLE	234 — угол с датчика угла наклона (TKAM) 12.
TKAM_13_ANGLE	235 — угол с датчика угла наклона (TKAM) 13.
TKAM_14_ANGLE	236 — угол с датчика угла наклона (TKAM) 14.
TKAM_15_ANGLE	237 — угол с датчика угла наклона (TKAM) 15.
TKAM_16_ANGLE	238 — угол с датчика угла наклона (TKAM) 16.
TKAM_1_ROLL	239 — крен с датчика угла наклона (TKAM) 1.
TKAM_2_ROLL	240 — крен с датчика угла наклона (TKAM) 2.
TKAM_3_ROLL	241 — крен с датчика угла наклона (TKAM) 3.
TKAM_4_ROLL	242 — крен с датчика угла наклона (TKAM) 4.
TKAM_5_ROLL	243 — крен с датчика угла наклона (TKAM) 5.
TKAM_6_ROLL	244 — крен с датчика угла наклона (TKAM) 6.
TKAM_7_ROLL	245 — крен с датчика угла наклона (TKAM) 7.
TKAM_8_ROLL	246 — крен с датчика угла наклона (TKAM) 8.
TKAM_9_ROLL	247 — крен с датчика угла наклона (TKAM) 9.
TKAM_10_ROLL	248 — крен с датчика угла наклона (TKAM) 10.

TKAM_11_ROLL	249 — крен с датчика угла наклона (TKAM) 11.
TKAM_12_ROLL	250 — крен с датчика угла наклона (TKAM) 12.
TKAM_13_ROLL	251 — крен с датчика угла наклона (TKAM) 13.
TKAM_14_ROLL	252 — крен с датчика угла наклона (TKAM) 14.
TKAM_15_ROLL	253 — крен с датчика угла наклона (TKAM) 15.
TKAM_16_ROLL	254 — крен с датчика угла наклона (TKAM) 16.
TKAM_1_PITCH	255 — тангаж с датчика угла наклона (TKAM) 1.
TKAM_2_PITCH	256 — тангаж с датчика угла наклона (TKAM) 2.
TKAM_3_PITCH	257 — тангаж с датчика угла наклона (TKAM) 3.
TKAM_4_PITCH	258 — тангаж с датчика угла наклона (TKAM) 4.
TKAM_5_PITCH	259 — тангаж с датчика угла наклона (TKAM) 5.
TKAM_6_PITCH	260 — тангаж с датчика угла наклона (TKAM) 6.
TKAM_7_PITCH	261 — тангаж с датчика угла наклона (TKAM) 7.
TKAM_8_PITCH	262 — тангаж с датчика угла наклона (TKAM) 8.
TKAM_9_PITCH	263 — тангаж с датчика угла наклона (TKAM) 9.
TKAM_10_PITCH	264 — тангаж с датчика угла наклона (TKAM) 10.
TKAM_11_PITCH	265 — тангаж с датчика угла наклона (TKAM) 11.
TKAM_12_PITCH	266 — тангаж с датчика угла наклона (TKAM) 12.
TKAM_13_PITCH	267 — тангаж с датчика угла наклона (TKAM) 13.
TKAM_14_PITCH	268 — тангаж с датчика угла наклона (TKAM) 14.
TKAM_15_PITCH	269 — тангаж с датчика угла наклона (TKAM) 15.
TKAM_16_PITCH	270 — тангаж с датчика угла наклона (TKAM) 16.
BRAKE_AIR_PRESSURE_1_SPN_1087	271 — давление в первом тормозном контуре, кПа.
BRAKE_AIR_PRESSURE_2_SPN_1088	272 — давление во втором тормозном контуре, кПа.
GROSS_VEHICLE_WEIGHT_SPN_1760	273 — общий вес автомобиля, кг.
LEVEL_A_IN_7	274 — напряжение аналогового входа 7, В.
LEVEL_A_IN_8	275 — напряжение аналогового входа 8, В.
LEVEL_NEAREST_UWB_ANCR_DIST	276 — расстояние до ближайшего якоря UWB, м.
LEVEL_AKB_CHARGE_PERCENT	277 — процент заряда батареи.
TKAM_1_RPM	278 — частота вращения датчика угла наклона (TKAM) 1, об/мин.
TKAM_2_RPM	279 — частота вращения датчика угла наклона (TKAM) 2, об/мин.
TKAM_3_RPM	280 — частота вращения датчика угла наклона (TKAM) 3, об/мин.
TKAM_4_RPM	281 — частота вращения датчика угла наклона (TKAM) 4, об/мин.
TKAM_5_RPM	282 — частота вращения датчика угла наклона (TKAM) 5, об/мин.
TKAM_6_RPM	283 — частота вращения датчика угла наклона (TKAM) 6, об/мин.
TKAM_7_RPM	284 — частота вращения датчика угла наклона (TKAM) 7, об/мин.

TKAM_8_RPM	285 — частота вращения датчика угла наклона (TKAM) 8, об/мин.
TKAM_9_RPM	286 — частота вращения датчика угла наклона (TKAM) 9, об/мин.
TKAM_10_RPM	287 — частота вращения датчика угла наклона (TKAM) 10, об/мин.
TKAM_11_RPM	288 — частота вращения датчика угла наклона (TKAM) 11, об/мин.
TKAM_12_RPM	289 — частота вращения датчика угла наклона (TKAM) 12, об/мин.
TKAM_13_RPM	290 — частота вращения датчика угла наклона (TKAM) 13, об/мин.
TKAM_14_RPM	291 — частота вращения датчика угла наклона (TKAM) 14, об/мин.
TKAM_15_RPM	292 — частота вращения датчика угла наклона (TKAM) 15, об/мин.
TKAM_16_RPM	293 — частота вращения датчика угла наклона (TKAM) 16, об/мин.
LEVEL_RAM_PARAM_1	294 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_2	295 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_3	296 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_4	297 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_5	298 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_6	299 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_7	300 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_8	301 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_9	302 — произвольный параметр для скриптов.
LEVEL_RAM_PARAM_10	303 — произвольный параметр для скриптов.

DiscrParamId

Дискретные параметры.

DISCR_INVALID_PARAM = 0	0 — не используется.
BRAKE_SWITCH_SPN_597	1 — педаль тормоза.
PARKING_BRAKE_SWITCH_SPN_70	2 — ручник.
OIL_PRESSURE_EMERGENCY_LAMP	3 — аварийная лампа давления масла.
CHECK_ENGINE_LAMP	4 — аварийная лампа неисправности двигателя.
ABS_FAULT_LAMP	5 — лампа неисправности ABS.
BRAKE_FAULT_LAMP	6 — лампа неисправности тормозов.
ESP_FAULT_LAMP	7 — лампа неисправности ESP.
CRUISE_CONTROL_SPN_595	8 — круиз-контроль.
CLUTCH_SWITCH_SPN_598	9 — педаль сцепления.
ACCEL_PEDAL_LOW_IDLE_SW_SPN_558	10 — режим холостого хода.
ACCEL_PEDAL_KICKDOWN_SW_SPN_559	11 — режим kickdown.
TORQUE_MODE_SPN_899	12 — режим крутящего момента двигателя.
PTO_STATE_SPN_976	13 — режим PTO.
CRUISE_CONTROL_STATE_SPN_527	14 — состояние круиз-контроля.
IGNITION	15 — зажигание.
IGNITION_KEY_IN_LOCK	16 — ключ в замке зажигания.
DYNAMIC_IGNITION_2	17 — динамическое зажигание 2.
WEBASTO	18 — webasto.
ENGINE_RUN	19 — двигатель работает.
ADD_ENGINE_RUN	20 — дополнительный двигатель работает.
READY_FOR_MOVE	21 — готов к движению.
ENGINE_ON_LNG	22 — двигатель работает на СПГ.
LEFT_FRONT_DOOR_OPEN	23 — левая передняя дверь открыта.
RIGHT_FRONT_DOOR_OPEN	24 — правая передняя дверь открыта.
LEFT_REAR_DOOR_OPEN	25 — левая задняя дверь открыта.
RIGHT_REAR_DOOR_OPEN	26 — правая задняя дверь открыта.
LUGGAGE_BOOT_DOOR_OPEN	27 — открыт багажник.
ENGINE_HOOD_OPEN	28 — открыт капот.
CHARGER_WIRE_CONNECTED	29 — провод зарядки подключен.
BATTERY_CHARGE	30 — зарядка аккумуляторной батареи включена.
AUTO_CLOSED	31 — автомобиль закрыт.
AUTO_CLOSED_BY_STANDARD_RC	32 — автомобиль закрыт при помощи штатного брелока.

STANDARD_ALARM_ON	33 — штатная сигнализация поставлена на охрану.
STANDARD_ALARM_EMULATION_ON	34 — эмуляция штатной сигнализации активирована.
STANDARD_RC_CL_SIG_SENT_ONCE	35 — сигнал закрытия с помощью заводского ПДУ был отправлен.
STANDARD_RC_OPENING_SIGNAL_SENT	36 — сигнал открытия с помощью заводского ПДУ был отправлен.
REPEAT_CLOSING_SIGNAL_SENT	37 — сигнал перепостановки был отправлен.
LUGGAGE_BOOT_OPENED_BY_RC	38 — багажник был открыт ПДУ.
CAN_SLEEP_MODE	39 — CAN-модуль в спящем режиме.
STANDARD_RC_CL_SIG_SNT_3_TIMES	40 — сигнал закрытия с помощью заводского ПДУ был отправлен трехкратно.
AGB_PARKING_ON	41 — АКПП в режиме «Парковка».
GB_REVERSE_ON	42 — КПП в режиме «Задний ход».
AGB_NEUTRAL_ON	43 — АКПП в режиме «Нейтраль».
AGB_MOVE_ON	44 — АКПП в режиме «Движение».
PARKING_LIGHTS	45 — парковочные огни включены.
LOW_BEAM_HEADLIGHTS	46 — ближний свет фар включен.
HIGH_BEAM_HEADLIGHTS	47 — дальний свет фар включен.
REAR_FOG_LIGHTS	48 — задние противотуманные фонари включены.
AIR_CONDITIONING	49 — кондиционер включен.
AUTO_RETARDER	50 — автоматический ретардер.
MANUAL_RETARDER	51 — ручной ретардер.
DRIVER_SEAT_BELT	52 — ремень водителя пристегнут.
FRONT_PASSENGER_SEAT_BELT	53 — ремень переднего пассажира пристегнут.
REAR_LEFT_PASSENGER_SEAT_BELT	54 — ремень заднего левого пассажира пристегнут.
REAR_RIGHT_PASSENGER_SEAT_BELT	55 — ремень заднего правого пассажира пристегнут.
REAR_CENTER_PASSENGER_SEAT_BELT	56 — ремень заднего центрального пассажира пристегнут.
FRONT_PASSENGER_SEAT_BELT_PR	57 — передний пассажирский ремень присутствует.
ESP_OFF	58 — ESP выключена.
STOP_LAMP	59 — лампа STOP.
COOLANT_EMERGENCY_LAMP	60 — лампа температуры/уровня воды.
BATTERY_LAMP	61 — индикатор отсутствия зарядки АКБ.
PARKING_BRAKE_LAMP	62 — индикатор системы стояночного тормоза.
AIRBAG_LAMP	63 — индикатор подушки безопасности.
EPS_FAULT_LAMP	64 — индикатор отказа EPS (электроусилитель руля).
WARNING_LAMP	65 — индикатор предупреждения.
EXTERNAL_LIGHTING_FAULT_LAMP	66 — индикатор неисправности внешних световых приборов.
TYRES_LOW_PRESSURE_LAMP	67 — индикатор низкого давления в шинах.
BRAKE_PADS_WEAR_LAMP	68 — индикатор износа тормозных колодок.

LOW_FUEL_LEVEL_LAMP	69 — индикатор низкого уровня топлива.
MAINTENANCE_LAMP	70 — индикатор наступления времени технического обслуживания.
GLOWPLUG_LAMP	71 — индикатор калильных свечей.
DPF_LAMP	72 — лампа DPF (сажевый фильтр, FAP).
EPC_LAMP	73 — индикатор EPC (электронный контроль мощности).
ENGINE_OIL_LOW_PRESSURE_LAMP	74 — индикатор низкого давления масла в двигателе.
ENGINE_OIL_HIGH_PRESSURE_LAMP	75 — индикатор низкого давления масла в двигателе.
COOLANT_LOW_LEVEL_LAMP	76 — индикатор низкого уровня охлаждающей жидкости.
HYDRO_FILTER_LAMP	77 — индикатор засорения фильтра масляной гидросистемы.
HYDRO_OIL_FILTER_LAMP	78 — индикатор засорения масляного фильтра гидросистемы.
HYDRO_LOW_PRESSURE_LAMP	79 — индикатор низкого давления в гидросистеме.
HYDRO_LOW_LEVEL_LAMP	80 — индикатор низкого уровня масла в гидросистеме.
HYDRO_HIGH_TEMPERATURE_LAMP	81 — индикатор высокой температуры в гидросистеме.
HYDRO_HIGH_LEVEL_LAMP	82 — индикатор перелива масла в баке в гидросистеме.
AIR_FILTER_LAMP	83 — индикатор засорения воздушного фильтра.
FUEL_FILTER_LAMP	84 — индикатор засорения топливного фильтра.
FUEL_WATER_LAMP	85 — индикатор присутствия воды в топливе.
BRAKE_FILTER_LAMP	86 — индикатор засорения фильтра тормозной системы.
CATALYST_OVERHEAT_LAMP	87 — индикатор перегрева катализатора.
AGRO_RIGHT_JOYSTICK_RIGHT	88 — правый джойстик вправо.
AGRO_RIGHT_JOYSTICK_LEFT	89 — правый джойстик влево.
AGRO_RIGHT_JOYSTICK_PUSH	90 — правый джойстик вперед.
AGRO_RIGHT_JOYSTICK_PULL	91 — правый джойстик назад.
AGRO_LEFT_JOYSTICK_RIGHT	92 — левый джойстик вправо.
AGRO_LEFT_JOYSTICK_LEFT	93 — левый джойстик влево.
AGRO_LEFT_JOYSTICK_PUSH	94 — левый джойстик вперед.
AGRO_LEFT_JOYSTICK_PULL	95 — левый джойстик назад.
AGRO_HYDRO_REAR_1	96 — первый задний гидропривод.
AGRO_HYDRO_REAR_2	97 — второй задний гидропривод.
AGRO_HYDRO_REAR_3	98 — третий задний гидропривод.
AGRO_HYDRO_REAR_4	99 — четвертый задний гидропривод.
AGRO_HYDRO_FRONT_1	100 — первый передний гидропривод.
AGRO_HYDRO_FRONT_2	101 — второй передний гидропривод.
AGRO_HYDRO_FRONT_3	102 — третий передний гидропривод.
AGRO_HYDRO_FRONT_4	103 — четвертый передний гидропривод.
AGRO_THREE_POINT_HITCH_FRONT	104 — передняя трехточечная система навески.

AGRO_THREE_POINT_HITCH_REAR	105 — задняя трехточечная система навески.
AGRO_PTO_FRONT_SPN_3452	106 — передний механизм отбора мощности.
AGRO_PTO_REAR_SPN_3453	107 — задний механизм отбора мощности.
AGRO_MOWING	108 — покос.
AGRO_THRESHING	109 — молотьяба.
AGRO_GRAIN_HOPPER_UNLOADING	110 — разгрузка зерна из бункера.
AGRO_GRAIN_HOPPER_100_LOAD	111 — зерновой бункер заполнен на 100 %.
AGRO_GRAIN_HOPPER_70_LOAD	112 — зерновой бункер заполнен на 70 %.
AGRO_GRAIN_HOPPER_OPEN	113 — зерновой бункер открыт.
AGRO_ULD_MECH_ACT_TUBE_AWAY	114 — привод выгрузного механизма при сложенной выгрузной трубе включен.
AGRO_CLEANING_FAN_CTRL_DIS	115 — управление вентилятором очистки отключено. 0b01 — отключено.
AGRO_THRESHING_DRUM_CTRL_DIS	116 — управление молотильным барабаном отключено. 0b01 — отключено.
AGRO_STRAW_WALKER_FAULT	117 — соломотряс забит.
AGRO_THRESHING_DRUM_EXC_CLR	118 — избыточный зазор под молотильным барабаном.
US_SALT_THROWER	119 — распылитель соли (песка).
US_REAGENTS_POURING	120 — разливка реагентов.
US_CONVEYOR_BELT	121 — конвейерный ремень.
US_SALT_THROWER_WHEEL_DRIVE	122 — привод колеса солеразбрасывателя.
US_BRUSH	123 — щетки.
US_VACUUM_CLEANER	124 — пылесос.
US_WATER_SUPPLY	125 — подача воды.
US_WASHING_MACHINE	126 — мойщик аппарата высокого давления.
US_WATER_PUMP	127 — насос подачи жидкости.
US_HOPPER_UNLOADING	128 — выгрузка из бункера.
US_SALT_LOW_LEVEL_LAMP	129 — индикатор низкого уровня соли (песка) в баке.
US_WATER_LOW_LEVEL_LAMP	130 — индикатор низкого уровня воды в баке.
US_REAGENTS_USAGE	131 — использование реагентов.
US_COMPRESSOR	132 — компрессор.
US_WATER_VALVE	133 — водяной клапан.
US_CABIN_MOVED_UP	134 — статус «Кабина перемещена вверх».
US_CABIN_MOVED_DOWN	135 — статус «Кабина перемещена вниз».
EDDP_ACCELERATION	136 — событие качества вождения: резкое ускорение.
EDDP_BREAKING	137 — событие качества вождения: резкое торможение.
EDDP_EXTRBREAKING	138 — событие качества вождения: экстренное торможение.
EDDP_RIGHTTURN	139 — событие качества вождения: резкий поворот направо.

EDDP_LEFTTURN	140 — событие качества вождения: резкий поворот налево.
EDDP_HOLE	141 — событие качества вождения: неровность дороги (яма).
EDDP_TILT	142 — событие качества вождения: опрокидывание.
EDDP_OVERTURN	143 — событие качества вождения: переворот.
EDDP_RESERVED3	144 — зарезервировано качество вождения.
EDDP_ANY_EVENT	145 — качество вождения: любое событие из контроля ускорений.
EDDP_SPEEDPOROG1	146 — превышен порог скорости 1.
EDDP_SPEEDPOROG2	147 — превышен порог скорости 2.
EDDP_SPEEDPOROG3	148 — превышен порог скорости 3.
MOT_MOTION_FROM_NAV	149 — признак движения по навигационному приемнику.
MOT_MOTION_FROM_ACCEL	150 — признак движения по акселерометру.
MOT_FAST_MOTION_FROM_NAV	151 — признак быстрого движения по навигационному приемнику.
TNS_CURRENT_STATE1	152 — текущее состояние подключения к серверу 1 (см. ServerConnectionStatus).
TNS_CURRENT_STATE2	153 — текущее состояние подключения к серверу 2 (см. ServerConnectionStatus).
TNS_CURRENT_STATE3	154 — текущее состояние подключения к серверу 3 (см. ServerConnectionStatus).
TNS_MAX_STATE1	155 — максимальное состояние подключения к серверу 1 (см. ServerConnectionStatus).
TNS_MAX_STATE2	156 — максимальное состояние подключения к серверу 2 (см. ServerConnectionStatus).
TNS_MAX_STATE3	157 — максимальное состояние подключения к серверу 3 (см. ServerConnectionStatus).
TKAM_1_OUT_1	158 — состояние выхода 1 ДУН 1.
TKAM_1_OUT_2	159 — состояние выхода 2 ДУН 1.
TKAM_2_OUT_1	160 — состояние выхода 1 ДУН 2.
TKAM_2_OUT_2	161 — состояние выхода 2 ДУН 2.
TKAM_3_OUT_1	162 — состояние выхода 1 ДУН 3.
TKAM_3_OUT_2	163 — состояние выхода 2 ДУН 3.
TKAM_4_OUT_1	164 — состояние выхода 1 ДУН 4.
TKAM_4_OUT_2	165 — состояние выхода 2 ДУН 4.
TKAM_5_OUT_1	166 — состояние выхода 1 ДУН 5.
TKAM_5_OUT_2	167 — состояние выхода 2 ДУН 5.
TKAM_6_OUT_1	168 — состояние выхода 1 ДУН 6.
TKAM_6_OUT_2	169 — состояние выхода 2 ДУН 6.
TKAM_7_OUT_1	170 — состояние выхода 1 ДУН 7.
TKAM_7_OUT_2	171 — состояние выхода 2 ДУН 7.

TKAM_8_OUT_1	172 — состояние выхода 1 ДУН 8.
TKAM_8_OUT_2	173 — состояние выхода 2 ДУН 8.
TKAM_9_OUT_1	174 — состояние выхода 1 ДУН 9.
TKAM_9_OUT_2	175 — состояние выхода 2 ДУН 9.
TKAM_10_OUT_1	176 — состояние выхода 1 ДУН 10.
TKAM_10_OUT_2	177 — состояние выхода 2 ДУН 10.
TKAM_11_OUT_1	178 — состояние выхода 1 ДУН 11.
TKAM_11_OUT_2	179 — состояние выхода 2 ДУН 11.
TKAM_12_OUT_1	180 — состояние выхода 1 ДУН 12.
TKAM_12_OUT_2	181 — состояние выхода 2 ДУН 12.
TKAM_13_OUT_1	182 — состояние выхода 1 ДУН 13.
TKAM_13_OUT_2	183 — состояние выхода 2 ДУН 13.
TKAM_14_OUT_1	184 — состояние выхода 1 ДУН 14.
TKAM_14_OUT_2	185 — состояние выхода 2 ДУН 14.
TKAM_15_OUT_1	186 — состояние выхода 1 ДУН 15.
TKAM_15_OUT_2	187 — состояние выхода 2 ДУН 15.
TKAM_16_OUT_1	188 — состояние выхода 1 ДУН 16.
TKAM_16_OUT_2	189 — состояние выхода 2 ДУН 16.
NAV_COORDS_VALID	190 — прием координат достоверен.
ID_IBUTTON	191 — считан идентификатор по iButton.
ID_BLE	192 — считан идентификатор по BLE.
ID_CAN	193 — считан идентификатор по CAN.
ID_MODBUS	194 — считан идентификатор по MODBUS.
POSITION_OF_DOORS_SPN_1821	195 — положение дверей.
RAMP_POSITION_SPN_1820	196 — рампа/лифт для коляски.
STATUS_2_OF_DOORS_SPN_3411	197 — статус дверей.
CURRENT_GEAR_SPN_523	198 — текущая передача.
FUEL_TYPE_SPN_5837	199 — используемое топливо.
OUTPUT_OK_STATE_1	200 — состояние выхода 1.
OUTPUT_OK_STATE_2	201 — состояние выхода 2.
OUTPUT_OK_STATE_3	202 — состояние выхода 3.
TKAM_1_EVENT_STATE	203 — состояние сработки события ДУН 1.
TKAM_2_EVENT_STATE	204 — состояние сработки события ДУН 2.
TKAM_3_EVENT_STATE	205 — состояние сработки события ДУН 3.
TKAM_4_EVENT_STATE	206 — состояние сработки события ДУН 4.
TKAM_5_EVENT_STATE	207 — состояние сработки события ДУН 5.

TKAM_6_EVENT_STATE	208 — состояние сработки события ДУН 6.
TKAM_7_EVENT_STATE	209 — состояние сработки события ДУН 7.
TKAM_8_EVENT_STATE	210 — состояние сработки события ДУН 8.
TKAM_9_EVENT_STATE	211 — состояние сработки события ДУН 9.
TKAM_10_EVENT_STATE	212 — состояние сработки события ДУН 10.
TKAM_11_EVENT_STATE	213 — состояние сработки события ДУН 11.
TKAM_12_EVENT_STATE	214 — состояние сработки события ДУН 12.
TKAM_13_EVENT_STATE	215 — состояние сработки события ДУН 13.
TKAM_14_EVENT_STATE	216 — состояние сработки события ДУН 14.
TKAM_15_EVENT_STATE	217 — состояние сработки события ДУН 15.
TKAM_16_EVENT_STATE	218 — состояние сработки события ДУН 16.
CAMERA_EVENT_STATE	219 — события камеры.
UWB_ANCHOR_DANGER_ZONE	220 — тег находится в опасной зоне (см. <code>DangerZoneType</code>).
MOT_STOP_FROM_NAV	221 — признак остановки по навигационному приемнику.
INT_BAT_CHARGING	222 — признак зарядки внутренней батареи.
LLS_1_ERRORS	223 — ошибки ДУТ 1.
LLS_2_ERRORS	224 — ошибки ДУТ 2.
LLS_3_ERRORS	225 — ошибки ДУТ 3.
LLS_4_ERRORS	226 — ошибки ДУТ 4.
LLS_5_ERRORS	227 — ошибки ДУТ 5.
LLS_6_ERRORS	228 — ошибки ДУТ 6.
LLS_7_ERRORS	229 — ошибки ДУТ 7.
LLS_8_ERRORS	230 — ошибки ДУТ 8.
TKAM_1_ROLL_CNTR	231 — количество оборотов, сделанное датчиком угла наклона (TKAM) 1.
TKAM_2_ROLL_CNTR	232 — количество оборотов, сделанное датчиком угла наклона (TKAM) 2.
TKAM_3_ROLL_CNTR	233 — количество оборотов, сделанное датчиком угла наклона (TKAM) 3.
TKAM_4_ROLL_CNTR	234 — количество оборотов, сделанное датчиком угла наклона (TKAM) 4.
TKAM_5_ROLL_CNTR	235 — количество оборотов, сделанное датчиком угла наклона (TKAM) 5.
TKAM_6_ROLL_CNTR	236 — количество оборотов, сделанное датчиком угла наклона (TKAM) 6.
TKAM_7_ROLL_CNTR	237 — количество оборотов, сделанное датчиком угла наклона (TKAM) 7.
TKAM_8_ROLL_CNTR	238 — количество оборотов, сделанное датчиком угла наклона (TKAM) 8.
TKAM_9_ROLL_CNTR	239 — количество оборотов, сделанное датчиком угла наклона (TKAM) 9.
TKAM_10_ROLL_CNTR	240 — количество оборотов, сделанное датчиком угла наклона (TKAM) 10.
TKAM_11_ROLL_CNTR	241 — количество оборотов, сделанное датчиком угла наклона (TKAM) 11.
TKAM_12_ROLL_CNTR	242 — количество оборотов, сделанное датчиком угла наклона (TKAM) 12.
TKAM_13_ROLL_CNTR	243 — количество оборотов, сделанное датчиком угла наклона (TKAM) 13.

TKAM_14_ROLL_CNTR	244 — количество оборотов, сделанное датчиком угла наклона (TKAM) 14.
TKAM_15_ROLL_CNTR	245 — количество оборотов, сделанное датчиком угла наклона (TKAM) 15.
TKAM_16_ROLL_CNTR	246 — количество оборотов, сделанное датчиком угла наклона (TKAM) 16.
DISCR_RAM_PARAM_1	247 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_2	248 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_3	249 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_4	250 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_5	251 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_6	252 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_7	253 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_8	254 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_9	255 — произвольный параметр для скриптов.
DISCR_RAM_PARAM_10	256 — произвольный параметр для скриптов.
ANY_POI_IN	257 — признак нахождения в любой контрольной точке.

GenericParamsId

Произвольные параметры

GENERIC_INVALID_PARAM = 0	0 — не используется.
GENERIC_MODBUS_PARAM_1	1 — произвольный параметр датчика Modbus 1.
GENERIC_MODBUS_PARAM_2	2 — произвольный параметр датчика Modbus 2.
GENERIC_MODBUS_PARAM_3	3 — произвольный параметр датчика Modbus 3.
GENERIC_MODBUS_PARAM_4	4 — произвольный параметр датчика Modbus 4.
GENERIC_MODBUS_PARAM_5	5 — произвольный параметр датчика Modbus 5.
GENERIC_MODBUS_PARAM_6	6 — произвольный параметр датчика Modbus 6.
GENERIC_MODBUS_PARAM_7	7 — произвольный параметр датчика Modbus 7.
GENERIC_MODBUS_PARAM_8	8 — произвольный параметр датчика Modbus 8.
GENERIC_MODBUS_PARAM_9	9 — произвольный параметр датчика Modbus 9.
GENERIC_MODBUS_PARAM_10	10 — произвольный параметр датчика Modbus 10.
GENERIC_MODBUS_PARAM_11	11 — произвольный параметр датчика Modbus 11.
GENERIC_MODBUS_PARAM_12	12 — произвольный параметр датчика Modbus 12.
GENERIC_MODBUS_PARAM_13	13 — произвольный параметр датчика Modbus 13.
GENERIC_MODBUS_PARAM_14	14 — произвольный параметр датчика Modbus 14.
GENERIC_MODBUS_PARAM_15	15 — произвольный параметр датчика Modbus 15.
GENERIC_MODBUS_PARAM_16	16 — произвольный параметр датчика Modbus 16.
GENERIC_MODBUS_PARAM_17	17 — произвольный параметр датчика Modbus 17.
GENERIC_MODBUS_PARAM_18	18 — произвольный параметр датчика Modbus 18.
GENERIC_MODBUS_PARAM_19	19 — произвольный параметр датчика Modbus 19.
GENERIC_MODBUS_PARAM_20	20 — произвольный параметр датчика Modbus 20.
GENERIC_MODBUS_PARAM_21	21 — произвольный параметр датчика Modbus 21.
GENERIC_MODBUS_PARAM_22	22 — произвольный параметр датчика Modbus 22.
GENERIC_MODBUS_PARAM_23	23 — произвольный параметр датчика Modbus 23.
GENERIC_MODBUS_PARAM_24	24 — произвольный параметр датчика Modbus 24.
GENERIC_MODBUS_PARAM_25	25 — произвольный параметр датчика Modbus 25.
GENERIC_MODBUS_PARAM_26	26 — произвольный параметр датчика Modbus 26.
GENERIC_MODBUS_PARAM_27	27 — произвольный параметр датчика Modbus 27.
GENERIC_MODBUS_PARAM_28	28 — произвольный параметр датчика Modbus 28.
GENERIC_MODBUS_PARAM_29	29 — произвольный параметр датчика Modbus 29.
GENERIC_MODBUS_PARAM_30	30 — произвольный параметр датчика Modbus 30.
GENERIC_MODBUS_PARAM_31	31 — произвольный параметр датчика Modbus 31.
GENERIC_MODBUS_PARAM_32	32 — произвольный параметр датчика Modbus 32.

GENERIC_MODBUS_PARAM_33	33 — произвольный параметр датчика Modbus 33.
GENERIC_MODBUS_PARAM_34	34 — произвольный параметр датчика Modbus 34.
GENERIC_MODBUS_PARAM_35	35 — произвольный параметр датчика Modbus 35.
GENERIC_MODBUS_PARAM_36	36 — произвольный параметр датчика Modbus 36.
GENERIC_MODBUS_PARAM_37	37 — произвольный параметр датчика Modbus 37.
GENERIC_MODBUS_PARAM_38	38 — произвольный параметр датчика Modbus 38.
GENERIC_MODBUS_PARAM_39	39 — произвольный параметр датчика Modbus 39.
GENERIC_MODBUS_PARAM_40	40 — произвольный параметр датчика Modbus 40.
GENERIC_MODBUS_PARAM_41	41 — произвольный параметр датчика Modbus 41.
GENERIC_MODBUS_PARAM_42	42 — произвольный параметр датчика Modbus 42.
GENERIC_MODBUS_PARAM_43	43 — произвольный параметр датчика Modbus 43.
GENERIC_MODBUS_PARAM_44	44 — произвольный параметр датчика Modbus 44.
GENERIC_MODBUS_PARAM_45	45 — произвольный параметр датчика Modbus 45.
GENERIC_MODBUS_PARAM_46	46 — произвольный параметр датчика Modbus 46.
GENERIC_MODBUS_PARAM_47	47 — произвольный параметр датчика Modbus 47.
GENERIC_MODBUS_PARAM_48	48 — произвольный параметр датчика Modbus 48.
GENERIC_MODBUS_PARAM_49	49 — произвольный параметр датчика Modbus 49.
GENERIC_MODBUS_PARAM_50	50 — произвольный параметр датчика Modbus 50.
GENERIC_MODBUS_PARAM_51	51 — произвольный параметр датчика Modbus 51.
GENERIC_MODBUS_PARAM_52	52 — произвольный параметр датчика Modbus 52.
GENERIC_MODBUS_PARAM_53	53 — произвольный параметр датчика Modbus 53.
GENERIC_MODBUS_PARAM_54	54 — произвольный параметр датчика Modbus 54.
GENERIC_MODBUS_PARAM_55	55 — произвольный параметр датчика Modbus 55.
GENERIC_MODBUS_PARAM_56	56 — произвольный параметр датчика Modbus 56.
GENERIC_MODBUS_PARAM_57	57 — произвольный параметр датчика Modbus 57.
GENERIC_MODBUS_PARAM_58	58 — произвольный параметр датчика Modbus 58.
GENERIC_MODBUS_PARAM_59	59 — произвольный параметр датчика Modbus 59.
GENERIC_MODBUS_PARAM_60	60 — произвольный параметр датчика Modbus 60.
GENERIC_MODBUS_PARAM_61	61 — произвольный параметр датчика Modbus 61.
GENERIC_MODBUS_PARAM_62	62 — произвольный параметр датчика Modbus 62.
GENERIC_MODBUS_PARAM_63	63 — произвольный параметр датчика Modbus 63.
GENERIC_MODBUS_PARAM_64	64 — произвольный параметр датчика Modbus 64.
GENERIC_MODBUS_PARAM_65	65 — произвольный параметр датчика Modbus 65.
GENERIC_MODBUS_PARAM_66	66 — произвольный параметр датчика Modbus 66.
GENERIC_MODBUS_PARAM_67	67 — произвольный параметр датчика Modbus 67.
GENERIC_MODBUS_PARAM_68	68 — произвольный параметр датчика Modbus 68.

GENERIC_MODBUS_PARAM_69	69 — произвольный параметр датчика Modbus 69.
GENERIC_MODBUS_PARAM_70	70 — произвольный параметр датчика Modbus 70.
GENERIC_MODBUS_PARAM_71	71 — произвольный параметр датчика Modbus 71.
GENERIC_MODBUS_PARAM_72	72 — произвольный параметр датчика Modbus 72.
GENERIC_MODBUS_PARAM_73	73 — произвольный параметр датчика Modbus 73.
GENERIC_MODBUS_PARAM_74	74 — произвольный параметр датчика Modbus 74.
GENERIC_MODBUS_PARAM_75	75 — произвольный параметр датчика Modbus 75.
GENERIC_MODBUS_PARAM_76	76 — произвольный параметр датчика Modbus 76.
GENERIC_MODBUS_PARAM_77	77 — произвольный параметр датчика Modbus 77.
GENERIC_MODBUS_PARAM_78	78 — произвольный параметр датчика Modbus 78.
GENERIC_MODBUS_PARAM_79	79 — произвольный параметр датчика Modbus 79.
GENERIC_MODBUS_PARAM_80	80 — произвольный параметр датчика Modbus 80.
GENERIC_MODBUS_PARAM_81	81 — произвольный параметр датчика Modbus 81.
GENERIC_MODBUS_PARAM_82	82 — произвольный параметр датчика Modbus 82.
GENERIC_MODBUS_PARAM_83	83 — произвольный параметр датчика Modbus 83.
GENERIC_MODBUS_PARAM_84	84 — произвольный параметр датчика Modbus 84.
GENERIC_MODBUS_PARAM_85	85 — произвольный параметр датчика Modbus 85.
GENERIC_MODBUS_PARAM_86	86 — произвольный параметр датчика Modbus 86.
GENERIC_MODBUS_PARAM_87	87 — произвольный параметр датчика Modbus 87.
GENERIC_MODBUS_PARAM_88	88 — произвольный параметр датчика Modbus 88.
GENERIC_MODBUS_PARAM_89	89 — произвольный параметр датчика Modbus 89.
GENERIC_MODBUS_PARAM_90	90 — произвольный параметр датчика Modbus 90.
GENERIC_MODBUS_PARAM_91	91 — произвольный параметр датчика Modbus 91.
GENERIC_MODBUS_PARAM_92	92 — произвольный параметр датчика Modbus 92.
GENERIC_MODBUS_PARAM_93	93 — произвольный параметр датчика Modbus 93.
GENERIC_MODBUS_PARAM_94	94 — произвольный параметр датчика Modbus 94.
GENERIC_MODBUS_PARAM_95	95 — произвольный параметр датчика Modbus 95.
GENERIC_MODBUS_PARAM_96	96 — произвольный параметр датчика Modbus 96.
GENERIC_MODBUS_PARAM_97	97 — произвольный параметр датчика Modbus 97.
GENERIC_MODBUS_PARAM_98	98 — произвольный параметр датчика Modbus 98.
GENERIC_MODBUS_PARAM_99	99 — произвольный параметр датчика Modbus 99.
GENERIC_MODBUS_PARAM_100	100 — произвольный параметр датчика Modbus 100.
GENERIC_CAN_PARAM_1	101 — произвольный параметр CAN 1.
GENERIC_CAN_PARAM_2	102 — произвольный параметр CAN 2.
GENERIC_CAN_PARAM_3	103 — произвольный параметр CAN 3.
GENERIC_CAN_PARAM_4	104 — произвольный параметр CAN 4.

GENERIC_CAN_PARAM_5	105 — произвольный параметр CAN 5.
GENERIC_CAN_PARAM_6	106 — произвольный параметр CAN 6.
GENERIC_CAN_PARAM_7	107 — произвольный параметр CAN 7.
GENERIC_CAN_PARAM_8	108 — произвольный параметр CAN 8.
GENERIC_CAN_PARAM_9	109 — произвольный параметр CAN 9.
GENERIC_CAN_PARAM_10	110 — произвольный параметр CAN 10.
GENERIC_CAN_PARAM_11	111 — произвольный параметр CAN 11.
GENERIC_CAN_PARAM_12	112 — произвольный параметр CAN 12.
GENERIC_CAN_PARAM_13	113 — произвольный параметр CAN 13.
GENERIC_CAN_PARAM_14	114 — произвольный параметр CAN 14.
GENERIC_CAN_PARAM_15	115 — произвольный параметр CAN 15.
GENERIC_CAN_PARAM_16	116 — произвольный параметр CAN 16.
GENERIC_CAN_PARAM_17	117 — произвольный параметр CAN 17.
GENERIC_CAN_PARAM_18	118 — произвольный параметр CAN 18.
GENERIC_CAN_PARAM_19	119 — произвольный параметр CAN 19.
GENERIC_CAN_PARAM_20	120 — произвольный параметр CAN 20.
GENERIC_CAN_PARAM_21	121 — произвольный параметр CAN 21.
GENERIC_CAN_PARAM_22	122 — произвольный параметр CAN 22.
GENERIC_CAN_PARAM_23	123 — произвольный параметр CAN 23.
GENERIC_CAN_PARAM_24	124 — произвольный параметр CAN 24.
GENERIC_CAN_PARAM_25	125 — произвольный параметр CAN 25.
GENERIC_CAN_PARAM_26	126 — произвольный параметр CAN 26.
GENERIC_CAN_PARAM_27	127 — произвольный параметр CAN 27.
GENERIC_CAN_PARAM_28	128 — произвольный параметр CAN 28.
GENERIC_CAN_PARAM_29	129 — произвольный параметр CAN 29.
GENERIC_CAN_PARAM_30	130 — произвольный параметр CAN 30.
GENERIC_CAN_PARAM_31	131 — произвольный параметр CAN 31.
GENERIC_CAN_PARAM_32	132 — произвольный параметр CAN 32.
GENERIC_CAN_PARAM_33	133 — произвольный параметр CAN 33.
GENERIC_CAN_PARAM_34	134 — произвольный параметр CAN 34.
GENERIC_CAN_PARAM_35	135 — произвольный параметр CAN 35.
GENERIC_CAN_PARAM_36	136 — произвольный параметр CAN 36.
GENERIC_CAN_PARAM_37	137 — произвольный параметр CAN 37.
GENERIC_CAN_PARAM_38	138 — произвольный параметр CAN 38.
GENERIC_CAN_PARAM_39	139 — произвольный параметр CAN 39.
GENERIC_CAN_PARAM_40	140 — произвольный параметр CAN 40.

GENERIC_CAN_PARAM_41	141 — произвольный параметр CAN 41.
GENERIC_CAN_PARAM_42	142 — произвольный параметр CAN 42.
GENERIC_CAN_PARAM_43	143 — произвольный параметр CAN 43.
GENERIC_CAN_PARAM_44	144 — произвольный параметр CAN 44.
GENERIC_CAN_PARAM_45	145 — произвольный параметр CAN 45.
GENERIC_CAN_PARAM_46	146 — произвольный параметр CAN 46.
GENERIC_CAN_PARAM_47	147 — произвольный параметр CAN 47.
GENERIC_CAN_PARAM_48	148 — произвольный параметр CAN 48.
GENERIC_CAN_PARAM_49	149 — произвольный параметр CAN 49.
GENERIC_CAN_PARAM_50	150 — произвольный параметр CAN 50.

ServerConnectionStatus

Статус подключения к серверу.

SCS_BEGINNING = 0	0 — начальное состояние подключения.
SCS_CLOSED = 1	1 — соединение закрыто.
SCS_CONNECTING = 2	2 — попытка подключения.
SCS_CONNECTED = 3	3 — соединение установлено.
SCS_DATA_SENT = 4	4 — данные на сервер отправлены.
SCS_ANSWER_OK = 5	5 — успешная передача данных на сервер.
SCS_ERROR_PASSWORD = 100	100 — пароль на сервере и контроллере не совпадает.
SCS_ERROR_NOT_SERVICED = 101	101 — контроллер не обслуживается на сервере.
SCS_ERROR_WRONG_CONFIGURATION = 102	102 — некорректная конфигурация сервера.
SCS_ERROR_WRONG_CHANNEL = 103	103 — передача данных по неправильному каналу.
SCS_ERROR_WRONG_PROTOCOL = 104	104 — протокол АвтоГРАФ (legacy) недоступен для комбинации данного сервера и контроллера.
SCS_ERROR_NOT_ANSWER_OK = 105	105 — нет ответа от сервера.

DangerZoneType

Типы зон в системе приближения.

DANGER_ZONE_SAFE = 0	0 — зеленая зона.
DANGER_ZONE_DANGER = 1	1 — красная зона.
DANGER_ZONE_WARNING = 2	2 — желтая зона.
DANGER_ZONE_UWB_DISABLED = 3	3 — UWB выключен.

Функции для работы с последовательными портами

Список функций	Описание
tkPortInit	Инициализация порта.
tkPortDeinit	Закрытие порта.
tkPortReadPackage	Чтение из порта пакетов, разделенных по временной задержке.
tkPortWrite	Запись данных в порт.
tkPortClearReadBuf	Очистка приемного буфера.

tkPortInit

Инициализация порта.

Для использования функции подключите файл:

```
#include <agSerial.inc>
```

Формат функции:

tkPortInit(index, speed, stopBits, parity)

Параметры:

index	Индекс открываемого порта: <ul style="list-style-type: none">• 0 — SERIAL_PORT_RS232 (RS-232);• 1 — SERIAL_PORT_RS485_1 (RS-485-1);• 2 — SERIAL_PORT_RS485_2 (RS-485-2).
speed	Скорость обмена данными: <ul style="list-style-type: none">• 0 — SB_AUTO (0 бит/с);• 9600 — SB_9600 (9600 бит/с);• 19200 — SB_19200 (19200 бит/с);• 38400 — SB_38400 (38400 бит/с);• 57600 — SB_57600 (57600 бит/с);• 115200 — SB_115200 (115200 бит/с).
stopBits	Количество стоп-битов: <ul style="list-style-type: none">• 0 — STOP_BITS_1 (1 стоп-бит);• 1 — STOP_BITS_2 (2 стоп-бита).
parity	Четность: <ul style="list-style-type: none">• 0 — PARITY_NONE (нет);• 1 — PARITY_EVEN (четный);• 2 — PARITY_ODD (нечетный).

Возвращаемое значение:

true — успешная инициализация, false — инициализация не удалась.

Примечание. После вызова функции `tkPortInit` работа с портом из основной прошивки прекращается. Чтобы работа с портом из основной прошивки продолжилась, нужно вызвать функцию `tkPortDeinit`.

Пример вызова функции:

```
tkPortInit(SERIAL_PORT_RS232, SB_115200, STOP_BITS_1, PARITY_NONE)
```

tkPortDeinit

Закрытие порта.

Для использования функции подключите файл:

```
#include <agSerial.inc>
```

Формат функции:

tkPortDeinit(index)

Параметры:

index	Индекс порта для закрытия: <ul style="list-style-type: none">• 0 — SERIAL_PORT_RS232 (RS-232);• 1 — SERIAL_PORT_RS485_1 (RS-485-1);• 2 — SERIAL_PORT_RS485_2 (RS-485-2).
--------------	--

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkPortDeinit(0);
```

tkPortReadPackage

Чтение из порта пакетов, разделенных по временной задержке.

Для использования функции подключите файл:

```
#include <agSerial.inc>
```

Функция читает последовательность байтов из указанного порта. Пакет определяется отсутствием данных в течение заданного интервала времени или размером буфера для получения пакета.

Формат функции:

```
tkPortReadPackage(index, buf, bufSize, firstByteTimeout, nextByteTimeout, bool:pack=true)
```

Параметры:

index	Индекс порта: <ul style="list-style-type: none">• 0 — SERIAL_PORT_RS232 (RS-232);• 1 — SERIAL_PORT_RS485_1 (RS-485-1);• 2 — SERIAL_PORT_RS485_2 (RS-485-2).
buf	Указатель на буфер для получения пакета.
bufSize	Размер передаваемого буфера в байтах.
firstByteTimeout	Время ожидания первого байта пакета, в миллисекундах.
nextByteTimeout	Время ожидания прихода bufSize байт, в миллисекундах.
pack	Приемный буфер запакован.

Возвращаемое значение:

Количество прочитанных байтов в пакете (0, если пакет не был получен).

Пример вызова функции:

```
new rxBuf{32};
new bytesRead = tkPortReadPackage(SERIAL_PORT_RS485_1, rxBuf, 32, 100, 100);
if (bytesRead > 0) {
    // Обработать rxBuf
}
```

tkPortWrite

Запись данных в порт.

Для использования функции подключите файл:

```
#include <agSerial.inc>
```

Функция передает указанный буфер данных в открытый последовательный порт.

Формат функции:

tkPortWrite(index, buf, bufSize, packed)

Параметры:

index	Индекс открытого порта: <ul style="list-style-type: none">• 0 — SERIAL_PORT_RS232 (RS-232);• 1 — SERIAL_PORT_RS485_1 (RS-485-1);• 2 — SERIAL_PORT_RS485_2 (RS-485-2).
buf[]	Указатель на буфер с данными для записи.
bufSize	Размер буфера в байтах.
packed	Указатель на то, запакован ли буфер с данными.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
new txBuf{4};  
tkPortWrite(SERIAL_PORT_RS485_1, txBuf, 4);
```

tkPortClearReadBuf

Очистка приемного буфера.

Для использования функции подключите файл:

```
#include <agSerial.inc>
```

Функция удаляет все принятые данные из буфера открытого последовательного порта.

Формат функции:

tkPortClearReadBuf(index)

Параметры:

index	Индекс открытого порта: <ul style="list-style-type: none">• 0 — SERIAL_PORT_RS232 (RS-232);• 1 — SERIAL_PORT_RS485_1 (RS-485-1);• 2 — SERIAL_PORT_RS485_2 (RS-485-2).
--------------	---

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkPortClearReadBuf(SERIAL_PORT_RS485_1);
```

Функции для работы с сокетами

Список функций	Описание
tkSocketOpen	Открытие сокета и подключение к серверу.
tkSocketSend	Отправка данных в открытый сокет.
tkSocketRecv	Получение данных из открытого сокета.
tkSocketClose	Закрытие сокета.

tkSocketOpen

Открытие сокета и подключение к серверу.

Для использования функции подключите файл:

```
#include <agSocket.inc>
```

Формат функции:

tkSocketOpen (address[], port, timeout)

Параметры:

address	Адрес сервера для подключения (доменное имя или IP-адрес).
port	Порт для подключения.
timeout	Время ожидания соединения.

Возвращаемое значение:

1 — подключение установлено, 0 — подключиться не удалось.

Пример вызова функции:

```
#define PORT 2225
#define TIMEOUT 4000
new address{} = "office.tk-chel.ru";
if ( !tkSocketOpen(address, PORT, TIMEOUT) ) {
    // обработка ошибки открытия сокета
}
```

tkSocketSend

Отправка данных в открытый сокет.

Для использования функции подключите файл:

```
#include <agSocket.inc>
```

Формат функции:

tkSocketSend (src[], bytesToSend, &bytesSent, timeout)

Параметры:

src	Массив данных для отправки.
bytesToSend	Количество байтов, которое нужно отправить.
bytesSent	Выходной параметр: фактически отправленное количество байтов.
timeout	Время ожидания операции (в миллисекундах).

Возвращаемое значение:

1 — данные успешно отправлены, 0 — ошибка отправки.

Пример вызова функции:

```
new buffer{} = "Hello, server!";
new sent = 0;
if ( !tkSocketSend(buffer, 14, sent, 5000) ) {
    // обработка ошибки отправки
}
```

tkSocketRecv

Получение данных из открытого сокета.

Для использования функции подключите файл:

```
#include <agSocket.inc>
```

Формат функции:

tkSocketRecv (dest[], bufSize, &bytesRcvd, timeout)

Параметры:

dest	Буфер для приема данных.
bufSize	Размер приемного буфера в байтах.
bytesRcvd	Выходной параметр: фактически полученное количество байтов.
timeout	Время ожидания операции (в миллисекундах).

Возвращаемое значение:

1 — данные успешно получены, 0 — ошибка получения.

Примечание. Функция завершится, если был заполнен весь буфер или истекло время ожидания операции.

Пример вызова функции:

```
new recvBuf{256};  
new received = 0;  
if ( tkSocketRecv(recvBuf, 256, received, 5000) ) {  
    // обработка полученных данных  
} else {  
    // обработка ошибки приема  
}
```

tkSocketClose

Закрытие сокета.

Для использования функции подключите файл:

```
#include <agSocket.inc>
```

Формат функции:

tkSocketClose ()

Параметры:

Нет.

Возвращаемое значение:

1 — сокет успешно закрыт, 0 — ошибка закрытия.

Пример вызова функции:

```
if (!tkSocketClose()) {  
    // обработка ошибки закрытия сокета  
}
```

Функции для работы с файлами в SPI памяти

Список функций	Описание
tkNextFileSPI	Получение списка файлов в заданной папке.
tkFileSizeSPI	Получение размера файла.
tkFileReadSPI	Чтение данных из файла с указанным смещением.
tkFileWriteSPI	Запись данных в файл с указанным смещением.
tkFileDeleteSPI	Удаление файла.
tkFileSendSPI	Отправка файла на сервер.
tkFileSendStatusSPI	Получение статуса отправки файла.
tkFileReadMCU	Чтение данных из файла (из флэш памяти микроконтроллера) с указанным смещением.

Список групп параметров	Описание
FRESULT	Возможные результаты операций.

tkNextFileSPI

Получение списка файлов в заданной папке.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkNextFileSPI (rootDir[], lastFile[], res[], resSize, &resultOut)

Параметры:

rootDir	Полный путь до папки, в которой производится поиск (относительно SPI).
lastFile	Последний файл, найденный в процессе поиска.
res	Массив, в который будут записаны путь и имя найденного файла (упакованная строка).
resSize	Размер массива res в байтах.
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

1 — файл найден, 0 — файлов больше нет.

Пример вызова функции:

```
new rootDir{} = "";  
new lastFile{} = "";  
new result{256};  
if ( tkNextFileSPI (rootDir, lastFile, result, 256) ) {  
    // обработка найденного файла  
}
```

tkFileSizeSPI

Получение размера файла.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileSizeSPI (filename[], &resultOut)

Параметры:

filename	Имя файла (относительно SPI).
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

Размер файла в байтах, -1 — файл не найден или ошибка.

Пример вызова функции:

```
new filename{} = "test.txt";  
new size = tkFileSizeSPI (filename);  
if ( size > 0 ) {  
    // обработка размера файла  
}
```

tkFileReadSPI

Чтение данных из файла с указанным смещением.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileReadSPI (filename[], buf[], bufSize, offset = 0, &resultOut)

Параметры:

filename	Имя файла (относительно SPI).
buf	Буфер для данных.
bufSize	Размер буфера (ограничен 1024 байтами).
offset	Смещение в файле (по умолчанию 0).
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

Количество прочитанных байт, 0 — ошибка чтения из файла.

Примечание. За один вызов функция может прочитать не более 1024 байт.

Пример вызова функции:

```
new filename{} = "data.bin";
new buffer{1024};
new bytesRead = tkFileReadSPI (filename, buffer, 1024, 0);
if ( bytesRead > 0 ) {
    // обработка прочитанных данных
}
```

tkFileWriteSPI

Запись данных в файл с указанным смещением.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileWriteSPI (filename[], buf[], bufSize, offset, maxFileLen, &resultOut)

Параметры:

filename	Имя файла (относительно SPI).
buf	Буфер с данными.
bufSize	Размер данных для записи (ограничен 1024 байтами).
offset	Смещение в файле.
maxFileLen	Размер создаваемого файла (ограничен размером общего доступного пространства для файлов на SPI Flash).
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

Количество записанных байт, 0 — ошибка записи в файл.

Примечание. Особенности SPI Flash: если файл еще не существует, то место для него заранее выделяется через *maxFileLen*.

Примечание. За один вызов функция может записать не более 1024 байт.

Пример вызова функции:

```
new filename{} = "output.txt";
new data{} = "Hello, World!";
new bytesWritten = tkFileWriteSPI (filename, data, 13, 0, 1024);
if ( bytesWritten > 0 ) {
    // данные успешно записаны
}
```

tkFileDeleteSPI

Удаление файла.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileDeleteSPI (filename[], &resultOut)

Параметры:

filename	Имя файла (относительно SPI).
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

1 — файл удален, 0 — ошибка удаления файла.

Примечание. Чтобы избежать исчерпания ресурсов перезаписи внутренней флеш-памяти, вызов функции `tkFileDeleteSPI` из Т.Скрипт ограничен по времени: не чаще 10 раз в минуту, 20 раз в час и 30 раз в день.

Пример вызова функции:

```
new filename{} = "temp.txt";  
if ( tkFileDeleteSPI (filename) ) {  
    // файл успешно удален  
} else {  
    // ошибка удаления файла  
}
```

tkFileSendSPI

Отправка файла на сервер.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileSendSPI (filename[], deleteAfter = 1)

Параметры:

filename	Имя файла (относительно SPI).
deleteAfter	1 — удалить файл после отправки, 0 — не удалять файл.

Возвращаемое значение:

1 — задача отправки поставлена, 0 — поставить задачу отправки не удалось.

Пример вызова функции:

```
new filename{} = "mydata.bin";  
if ( tkFileSendSPI (filename, 1) ) {  
    // задача отправки поставлена  
}
```

tkFileSendStatusSPI

Получение статуса отправки файла.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileSendStatusSPI (filename[], outStatus[.state, .filesize, .progress])

Параметры:

filename	Имя файла (относительно SPI).
outStatus	Массив структур, в который будет записан статус: <ul style="list-style-type: none">• .state — текущее состояние отправки:<ul style="list-style-type: none">• 0 — нет файлов на отправку;• 1 — есть файлы на отправку;• 3 — идет передача файла на сервер;• 4 — передача файла на сервер успешно завершена;• 5 — передача файла на сервер завершена с ошибкой.• .filesize — размер файла в байтах.• .progress — размер уже отправленного файла в процентах.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
new filename{} = "mydata.bin";
new status[.state, .filesize, .progress];

tkFileSendStatusSPI(filename, status);
printf("state=%d filesize=%d progress=%d%%\r\n", status.state, status.filesize, status.progress);
```

tkFileReadMCU

Чтение данных из файла (из флэш памяти микроконтроллера) с указанным смещением.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileReadMCU (filename[], buf[], bufSize, offset = 0, &resultOut)

Параметры:

filename	Имя файла (относительно MCU).
buf	Буфер для данных.
bufSize	Размер буфера (ограничен 1024 байтами).
offset	Смещение в файле (по умолчанию 0).
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

Количество прочитанных байт, 0 — ошибка чтения из файла.

Примечание. За один вызов функция может прочитать не более 1024 байт.

Пример вызова функции:

```
new filename{} = "data.bin";
new buffer{1024};
new bytesRead = tkFileReadMCU (filename, buffer, 1024, 0);
if ( bytesRead > 0 ) {
    // обработка прочитанных данных
}
```

FRESULT

Возможные результаты операций.

FR_OK = 0	0 — Успешно.
FR_DISK_ERR	1 — Возникла критическая ошибка на низком уровне дискового ввода-вывода.
FR_INT_ERR	2 — Сбой утверждения (Assertion failed).
FR_NOT_READY	3 — Физический диск не может работать.
FR_NO_FILE	4 — Файл не найден.
FR_NO_PATH	5 — Путь не найден.
FR_INVALID_NAME	6 — Неверный формат имени пути.
FR_DENIED	7 — Доступ запрещен или каталог заполнен.
FR_EXIST	8 — Доступ запрещен.
FR_INVALID_OBJECT	9 — Неверный объект файла/каталога.
FR_WRITE_PROTECTED	10 — Физический диск защищен от записи.
FR_INVALID_DRIVE	11 — Неверный номер логического диска.
FR_NOT_ENABLED	12 — Для тома нет рабочей области.
FR_NO_FILESYSTEM	13 — Нет действительного тома FAT.
FR_MKFS_ABORTED	14 — Выполнение функции <code>f_mkfs()</code> прервано из-за проблемы.
FR_TIMEOUT	15 — Не удалось получить доступ к тому в течение заданного периода.
FR_LOCKED	16 — Операция отклонена в соответствии с политикой общего доступа к файлам.
FR_NOT_ENOUGH_CORE	17 — Не удалось выделить рабочий буфер для длинных имен файлов (LFN).
FR_TOO_MANY_OPEN_FILES	18 — Количество открытых файлов > <code>FF_FS_LOCK</code> .
FR_INVALID_PARAMETER	19 — Указанный параметр недействителен.

Функции для работы с файлами в SD/RAM памяти

Следующие команды относятся к работе с файловой системой на SD-карте и во внутренней оперативной памяти устройства (RAM). Если карта памяти установлена, то функции будут работать с ней. Если карта памяти не установлена, то работа с файловой системой будет осуществляться в RAM. Обратите внимание, что при потере питания RAM очищается. Проверка наличия карты памяти осуществляется при подаче питания на устройство.

Список функций	Описание
<u>tkNextFile</u>	Получение списка файлов в заданной папке.
<u>tkNextDir</u>	Получение списка папок, вложенных в заданную папку.
<u>tkFileSize</u>	Получение размера файла.
<u>tkFileRead</u>	Чтение данных из файла.
<u>tkFileWrite</u>	Запись данных в файл.
<u>tkFileDelete</u>	Удаление файла.
<u>tkFileSend</u>	Отправка файла на сервер.
<u>tkFileSendStatus</u>	Получение статуса отправки файла.
<u>tkIsSdCardInserted</u>	Проверка наличия карты памяти в устройстве.

tkNextFile

Получение списка файлов в заданной папке.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkNextFile (rootDir[], lastFile[], res[], resSize, &resultOut)

Параметры:

rootDir	Полный путь до папки, в которой производится поиск.
lastFile	Последний файл, найденный в процессе поиска.
res	Массив, в который будут записаны путь и имя найденного файла (упакованная строка).
resSize	Размер массива res в байтах.
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

1 — файл найден, 0 — файлов больше нет.

Пример вызова функции:

```
new rootDir{} = "";  
new lastFile{} = "";  
new result{256};  
if ( tkNextFile (rootDir, lastFile, result, 256) ) {  
    // обработка найденного файла  
}
```

tkNextDir

Получение списка папок, вложенных в заданную папку.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkNextDir (rootDir[], lastDir[], res[], resSize, &resultOut)

Параметры:

rootDir	Полный путь до папки, в которой производится поиск.
lastDir	Последняя папка, найденная в процессе поиска.
res	Массив, в который будут записаны путь и имя найденной папки (упакованная строка).
resSize	Размер массива res в байтах.
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

1 — папка найдена, 0 — папок больше нет.

Пример вызова функции:

```
new rootDir{} = "";  
new lastDir{} = "";  
new result{256};  
if ( tkNextDir (rootDir, lastDir, result, 256) ) {  
    // обработка найденной папки  
}
```

tkFileSize

Получение размера файла.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileSize (filename[], &resultOut)

Параметры:

filename	Имя файла.
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

Размер файла в байтах, -1 — файл не найден или ошибка.

Пример вызова функции:

```
new filename{} = "test.txt";  
new size = tkFileSize (filename);  
if ( size > 0 ) {  
    // обработка размера файла  
}
```

tkFileRead

Чтение данных из файла.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileRead (filename[], buf[], bufSize, offset = 0, &resultOut)

Параметры:

filename	Полный или относительный путь.
buf	Буфер для данных.
bufSize	Размер буфера (ограничен 1024 байтами).
offset	Смещение в файле (по умолчанию 0).
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

Количество прочитанных байт, 0 — ошибка чтения файла.

Примечание. За один вызов функция может прочитать не более 1024 байт.

Пример вызова функции:

```
new filename{} = "data.bin";
new buffer{1024};
new bytesRead = tkFileRead (filename, buffer, 1024, 0);
if ( bytesRead > 0 ) {
    // обработка прочитанных данных
}
```

tkFileWrite

Запись данных в файл.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileWrite (filename[], buf[], bufSize, offset, &resultOut)

Параметры:

filename	Полный или относительный путь.
buf	Буфер с данными.
bufSize	Размер данных для записи (ограничен 1024 байтами).
offset	Смещение в файле.
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

Количество записанных байт, 0 — ошибка записи в файл.

Примечание. За один вызов функция может записать не более 1024 байт.

Пример вызова функции:

```
new filename{} = "output.txt";  
new data{} = "Hello, World!";  
new bytesWritten = tkFileWrite (filename, data, 13, 0);  
if ( bytesWritten > 0 ) {  
    // данные успешно записаны  
}
```

tkFileDelete

Удаление файла.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileDelete (filename[], &resultOut)

Параметры:

filename	Имя файла.
resultOut	Переменная, в которую будет записан результат операции (см. FRESULT). Необязательный параметр.

Возвращаемое значение:

1 — файл удален, 0 — ошибка удаления файла.

Пример вызова функции:

```
new filename{} = "temp.txt";  
if ( tkFileDelete (filename) ) {  
    // файл успешно удален  
} else {  
    // ошибка удаления файла  
}
```

tkFileSend

Отправка файла на сервер.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileSend (filename[], deleteAfter = 1)

Параметры:

filename	Имя файла.
deleteAfter	1 — удалить файл после отправки, 0 — не удалять файл.

Возвращаемое значение:

1 — задача отправки поставлена, 0 — поставить задачу отправки не удалось.

Пример вызова функции:

```
new filename{} = "mydata.bin";  
if ( tkFileSend (filename, 1) ) {  
    // задача отправки поставлена  
}
```

tkFileSendStatus

Получение статуса отправки файла.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkFileSendStatus (filename[], outStatus[.state, .filesize, .progress])

Параметры:

filename	Имя файла.
outStatus	Массив структур, в который будет записан статус: <ul style="list-style-type: none">• .state — текущее состояние отправки:<ul style="list-style-type: none">• 0 — нет файлов на отправку;• 1 — есть файлы на отправку;• 3 — идет передача файла на сервер;• 4 — передача файла на сервер успешно завершена;• 5 — передача файла на сервер завершена с ошибкой.• .filesize — размер файла в байтах.• .progress — размер уже отправленного файла в процентах.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
new filename{} = "mydata.bin";
new status[.state, .filesize, .progress];

tkFileSendStatus(filename, status);
printf("state=%d filesize=%d progress=%d%%\r\n", status.state, status.filesize, status.progress);
```

tkIsSdCardInserted

Проверка наличия карты памяти в устройстве.

Для использования функции подключите файл:

```
#include <agFile.inc>
```

Формат функции:

tkIsSdCardInserted()

Параметры:

Нет.

Возвращаемое значение:

1 — карта памяти установлена, 0 — карта памяти не установлена.

Пример вызова функции:

```
if (tkIsSdCardInserted())  
{  
    printf("SD-Card is inserted\r\n");  
}
```

Функции для работы с BLE

Данные функции доступны на прошивке BLE-чипа версии AGBT-01.14 и выше. Для проверки версии прошивки и ее обновления используйте следующие управляющие команды:

- GBLEVERSION — запрос версии прошивки BLE-чипа.
- BLEUPDATE=1 — обновление BLE-чипа на последнюю стабильную версию.

Список функций	Описание
tkBleStart	Запуск BLE в режиме работы скриптов.
tkBleAddAdrrsFromConf	Добавление в белый список адресов из конфигурации устройства.
tkBleAddAddrInWhiteList	Добавление MAC-адреса в белый список.
tkBleAddNameInWhiteList	Добавление маски имени в белый список.
tkBleClearWhiteList	Очистка белого списка устройств.
tkBleReceive	Прием Bluetooth-сообщения.
tkBleWriteRecord	Запись Bluetooth-данных в бинарный файл устройства.
tkBleSetRecordPeriod	Установка периода записи Bluetooth-данных.

tkBleStart

Запуск BLE в режиме работы скриптов.

Для использования функции подключите файл:

```
#include <agBle.inc>
```

Формат функции:

```
tkBleStart()
```

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkBleStart();
```

tkBleAddAdrsFromConf

Добавление в белый список адресов из конфигурации устройства.

Для использования функции подключите файл:

```
#include <agBle.inc>
```

Формат функции:

```
tkBleAddAdrsFromConf()
```

Возвращаемое значение:

Количество добавленных адресов.

Пример вызова функции:

```
new addr_counter = tkBleAddAdrsFromConf();
```

Примечание. Команда добавит датчики, адреса которых записаны в конфигурацию устройства.
(SCRIPTCONFSTRING=MAC1:AABBCCDDEE01,MAC2:AABBCCDDEE02,...,MAC8:AABBCCDDEE08;)

tkBleAddAddrInWhiteList

Добавление MAC-адреса в белый список.

Для использования функции подключите файл:

```
#include <agBle.inc>
```

Формат функции:

```
tkBleAddAddrInWhiteList(addr[BLEADDR])
```

Параметры:

addr	MAC-адрес устройства.
-------------	-----------------------

Возвращаемое значение:

Нет.

Пример вызова функции:

```
new mac_addr[BLEADDR];  
mac_addr.addr{0} = 0x1A;  
mac_addr.addr{1} = 0x2B;  
mac_addr.addr{2} = 0x3C;  
mac_addr.addr{3} = 0x4D;  
mac_addr.addr{4} = 0x5E;  
mac_addr.addr{5} = 0x6F;  
tkBleAddAddrInWhiteList(mac_addr);
```

Примечание. Доступно добавление до 8 адресов.

tkBleAddNameInWhiteList

Добавление маски имени в белый список.

Для использования функции подключите файл:

```
#include <agBle.inc>
```

Формат функции:

```
tkBleAddNameInWhiteList(const name[])
```

Параметры:

name	Маска имени устройства (строка).
------	----------------------------------

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkBleAddNameInWhiteList("abc");
```

Примечание. Доступно добавление до 8 масок.

Примечание. Чтобы фильтр сработал, рекламный пакет должен удовлетворять следующим условиям:

1. Он должен содержать структуру данных с типом `BLE_GAP_COMPLETE_LOCAL_NAME (0x09)`.
2. Значение в этой структуре должно начинаться с заданной маски (например, пакет с именем «abc123» будет принят по маске «abc»).

tkBleClearWhiteList

Очистка белого списка устройств.

Для использования функции подключите файл:

```
#include <agBle.inc>
```

Формат функции:

```
tkBleClearWhiteList()
```

Возвращаемое значение:

Нет.

Пример вызова функции:

```
tkBleClearWhiteList();
```

tkBleReceive

Прием Bluetooth-сообщения.

Для использования функции подключите файл:

```
#include <agBle.inc>
```

Формат функции:

```
tkBleReceive(msg[BLEMSG])
```

Параметры:

msg	Полученное сообщение.
------------	-----------------------

Возвращаемое значение:

Принято новое сообщение (1 — сообщение получено, 0 — сообщение не получено).

Пример вызова функции:

```
new ble_msg[BLEMSG];  
tkBleReceive(ble_msg);
```

tkBleWriteRecord

Запись Bluetooth-данных в бинарный файл устройства.

Для использования функции подключите файл:

```
#include <agBle.inc>
```

Формат функции:

```
tkBleWriteRecord(addr[BLEADDR], data{}, len)
```

Параметры:

addr	MAC-адрес устройства.
data	Данные.
len	Длина данных.

Возвращаемое значение:

Статус записи данных (1 — данные записаны, 0 — данные не записаны).

Пример вызова функции:

```
new ble_msg[BLEMSG];
if (tkBleReceive(ble_msg))
{
    tkBleWriteRecord(ble_msg.addr, ble_msg.data, ble_msg.len);
}
```

Примечание. Тип записи — 0xFA0001.

Примечание. Работает только для датчиков, адреса которых записаны в конфигурацию устройства и добавлены с помощью функции `tkBleAddAddrsFromConf()`.

tkBleSetRecordPeriod

Установка периода записи Bluetooth-данных.

Для использования функции подключите файл:

```
#include <agBle.inc>
```

Формат функции:

tkBleSetRecordPeriod(period)

Параметры:

period	Период записи, 5...300 с.
---------------	---------------------------

Возвращаемое значение:

Нет.

Пример вызова функции:

```
new period = 10;  
tkBleSetRecordPeriod(period);
```

Стандартные функции Pawn

Список функций	Описание
<u>printf</u>	Форматированный вывод текста в консоль сервера/клиента.
<u>heapspace</u>	Получение информации о количестве свободного пространства в секции стека/кучи скрипта.
<u>funcidx</u>	Проверка существования public-функции в скрипте.
<u>numargs</u>	Получение количества аргументов, переданных в текущую функцию.
<u>getarg</u>	Получение значения аргумента функции по его индексу.
<u>setarg</u>	Установка значения аргумента функции по указанному индексу.
<u>tolower</u>	Преобразование символа в нижний регистр.
<u>toupper</u>	Преобразование символа в верхний регистр.
<u>swapchars</u>	Перестановка байтов ячейки в обратном порядке.
<u>random</u>	Генерация псевдослучайного числа в заданном диапазоне.
<u>min</u>	Определение наименьшего из двух чисел.
<u>max</u>	Определение наибольшего из двух чисел.
<u>clamp</u>	Приведение числа к указанному диапазону.

printf

Форматированный вывод текста в консоль сервера/клиента.

Для использования функции подключите файл:

```
#include <console.inc>
```

Функция позволяет форматировать данные в строку (преобразовывать данные в строковый эквивалент и вставлять их в форматный текст) и выводить получившийся текст в консоль и в лог сервера.

Формат функции:

```
printf(const format[], {Fixed,}_:...)
```

Параметры:

format	Строка формата, определяющая вывод (аналогично стандартной printf в C).
...	Переменное количество аргументов, подставляемых в строку формата.

Возвращаемое значение:

Нет.

Спецификаторы формата:

- %d, %i — целое число (integer).
- %f — число с плавающей точкой (float).
- %s — строка (string).
- %x, %X — шестнадцатеричное число.
- %c — символ (character).
- %% — символ процента.

Пример вызова функции:

```
new hour, minute, second;  
gettime(hour, minute, second);  
printf("Текущее время: %d:%02d:%02d", hour, minute, second);
```

Примечание. В Rawn нет разницы между %d и %i.

Примечание. Для вывода чисел с плавающей точкой требуется явное приведение: `printf("%f", float(value))`.

Примечание. Максимальная длина выводимой строки — 1024 символа.

heapspace

Получение информации о количестве свободного пространства в секции стека/кучи скрипта.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

```
heapspace()
```

Возвращаемое значение:

Размер свободного пространства (в байтах) в секции стека/кучи.

Пример вызова функции:

```
new heap_free = heapspace();  
printf("В секции стека/кучи свободно %d байт", heap_free);
```

funcidx

Проверка существования public-функции в скрипте.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

```
funcidx(const name[])
```

Параметры:

name	Название функции.
------	-------------------

Возвращаемое значение:

Индекс функции в таблице public-функций. И -1, если функции с указанным названием не существует.

Пример вызова функции:

```
new idx = funcidx("MyFunction");  
// Вывод: "Функция MyFunction() существует (ID 0)"  
printf("Функция MyFunction() %существует (ID %d)", (idx == -1) ? "не " : "", idx);
```

Примечание. Проверяемая функция должна быть объявлена с атрибутом public либо ее название должно начинаться с символа «@».

Примечание. Индексы public-функций начинаются с 0.

numargs

Получение количества аргументов, переданных в текущую функцию.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

numargs()

Возвращаемое значение:

Количество аргументов, переданных в текущую функцию (0, если аргументов нет).

Пример вызова функции:

```
new n = numargs();  
printf("Количество аргументов: %d", n);  
for (new i = 0; i < n; ++i)  
    printf(" %d", getarg(i));
```

getarg

Получение значения аргумента функции по его индексу.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

```
getarg(arg, index=0)
```

Параметры:

arg	Индекс аргумента (начиная с 0).
index	Индекс элемента, если указанный аргумент является массивом (опционально, по умолчанию 0).

Возвращаемое значение:

Значение запрошенного аргумента в виде целого числа.

Пример вызова функции:

```
PrintArray(a[], size = sizeof(a))
{
    printf("Размер массива: %d", size);
    // Выведем все элементы массива
    for (new i = 0; i < size; ++i)
    {
        // То же самое, что и "printf(" %d", a[i]);"
        // На этот раз мы работаем с массивом, поэтому параметр index нужно указывать
        printf(" %d", getarg(0, i));
    }
}
```

setarg

Установка значения аргумента функции по указанному индексу.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

getarg(arg, index=0, value)

Параметры:

arg	Индекс аргумента (начиная с 0).
index	Индекс элемента, если указанный аргумент является массивом (параметр можно пропустить с помощью знака «_»).
value	Значение, которое нужно сохранить в аргументе.

Возвращаемое значение:

1 — значение установлено, 0 — указан неправильный индекс аргумента.

Пример вызова функции:

```
// Увеличим значение всех аргументов функции на 1
// Обратите внимание: в вызове setarg() параметр index пропущен (знак "_")
for (new i = 0, n = numargs(); i < n; ++i)
    setarg(i, _, getarg(i) + 1);
```

tolower

Преобразование символа в нижний регистр.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

tolower(c)

Параметры:

c	Символ, который нужно преобразовать в нижний регистр.
---	---

Возвращаемое значение:

Аналог символа **c** в нижнем регистре, если таковой существует. Иначе — исходное значение **c**.

Пример вызова функции:

```
static string[] = "THE QUICK FOX JUMPS OVER THE LAZY DOG";  
for (new i = 0; i < sizeof(string); ++i)  
    string[i] = tolower(string[i]);  
// Вывод: "the quick fox jumps over the lazy dog"  
print(string);
```

Примечание. Функция работает только с латинскими символами (A–Z).

Примечание. Функция не изменяет символы в нижнем регистре.

Примечание. Функция возвращает оригинальный символ, если он не является буквой верхнего регистра.

toupper

Преобразование символа в верхний регистр.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

toupper(c)

Параметры:

c	Символ, который нужно преобразовать в верхний регистр.
---	--

Возвращаемое значение:

Аналог символа **c** в нижнем регистре, если таковой существует. Иначе — исходное значение **c**.

Пример вызова функции:

```
static string[] = "the quick fox jumps over the lazy dog";  
for (new i = 0; i < sizeof(string); ++i)  
    string[i] = toupper(string[i]);  
// Вывод: "THE QUICK FOX JUMPS OVER THE LAZY DOG"  
print(string);
```

Примечание. Функция работает только с латинскими символами (A–Z).

Примечание. Функция не изменяет символы в верхнем регистре.

Примечание. Функция возвращает оригинальный символ, если он не является буквой нижнего регистра.

swapchars

Перестановка байтов ячейки в обратном порядке.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

swapchars(c)

Параметры:

c	Значение, байты которого нужно обменять местами.
---	--

Возвращаемое значение:

Значение аргумента c с байтами, переставленными в обратном порядке.

Пример вызова функции:

```
// Вывод: "swapchars(0x11223344): 0x44332211"  
new n = 0x11223344;  
printf("swapchars(0x%08x): 0x%08x", n, swapchars(n));
```

random

Генерация псевдослучайного числа в заданном диапазоне.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

random(max)

Параметры:

max	Верхняя граница диапазона для случайного числа.
-----	---

Возвращаемое значение:

Псевдослучайное число от 0 до (max – 1).

Пример вызова функции:

```
new r = random(4);
switch (r)
{
  case 0:
    print("Ноль");
  case 1:
    print("Один");
  case 2:
    print("Два");
  default:
    printf("%d", r);
}
```

Примечание. Если параметр **max** равен нулю, то функция примет это как отсутствие границ диапазона, в результате чего сгенерирует число от 0 до максимального значения, которое может вместить ячейка Pawn (*cellmax*).

Примечание. Функция работает на основе линейного конгруэнтного метода с диапазоном и периодом равными 2^{31} . Генераторы псевдослучайных чисел на основе этого метода подвержены серийной корреляции и непригодны для решения задач, требующих высокого качества случайности чисел (например, в криптографии), однако подходят для многих других задач.

min

Определение наименьшего из двух чисел.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

min(value1, value2)

Параметры:

value1	Первое число для сравнения (integer, float).
value2	Второе число для сравнения (integer, float).

Возвращаемое значение:

Наименьшее значение между **value1** и **value2**.

Пример вызова функции:

```
// Вывод: "Из чисел 3 и 4 наименьшее - 3"  
new a = 3, b = 4;  
printf("Из чисел %d и %d наименьшее - %d", a, b, min(a, b));
```

max

Определение наибольшего из двух чисел.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

max(value1, value2)

Параметры:

value1	Первое число для сравнения (integer, float).
value2	Второе число для сравнения (integer, float).

Возвращаемое значение:

Наибольшее значение между **value1** и **value2**.

Пример вызова функции:

```
// Вывод: "Из чисел 3 и 4 наибольшее - 4"  
new a = 3, b = 4;  
printf("Из чисел %d и %d наибольшее - %d", a, b, max(a, b));
```

clamp

Приведение числа к указанному диапазону.

Для использования функции подключите файл:

```
#include <core.inc>
```

Формат функции:

clamp(value, min=cellmin, max=cellmax)

Параметры:

value	Ограничиваемое значение (integer, float).
min	Нижняя граница диапазона (по умолчанию cellmin).
max	Верхняя граница диапазона (по умолчанию cellmax).

Возвращаемое значение:

Значение аргумента **value**, если оно находится в диапазоне от **min** до **max**.
Если **value** < **min**, то возвращает значение **min**. Иначе (**value** > **max**) — значение **max**.

Пример вызова функции:

```
new limited = clamp(150, 0, 100); // Вернет 100  
new Float:angle = clamp(450.0, 0.0, 360.0); // Вернет 360.0  
new autoClamped = clamp(9999); // Вернет cellmax
```

Стандартные функции Rown для чисел с фиксированной запятой

Список функций	Описание
<u>fixed</u>	Преобразование числа в формат с фиксированной запятой.
<u>strfixed</u>	Создание неизменяемой строки с фиксированным содержимым.
<u>fmul</u>	Умножение чисел с фиксированной запятой.
<u>fdiv</u>	Деление числа с фиксированной запятой.
<u>ffract</u>	Извлечение дробной части числа с фиксированной запятой.
<u>fround</u>	Округление числа с фиксированной запятой согласно указанному методу.
<u>fpower</u>	Возведение числа с фиксированной запятой в указанную степень.
<u>fsqroot</u>	Вычисление квадратного корня из числа с фиксированной запятой.
<u>fabs</u>	Получение абсолютного значения числа с фиксированной запятой.

fixed

Преобразование числа в формат с фиксированной запятой.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

Fixed:fixed(value)

Параметры:

value	Число для преобразования (целое, дробное или строка).
-------	---

Возвращаемое значение:

Значение с фиксированной запятой.

Пример вызова функции:

```
new Fixed:a = Fixed:fixed(10); // Целое число  
new Fixed:b = Fixed:fixed(3.14); // Дробное число  
new Fixed:c = Fixed:fixed("2.718"); // Строковое представление
```

strfixed

Создание неизменяемой строки с фиксированным содержимым.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

Fixed:strfixed(const string[])

Параметры:

string[]	Строка для фиксации (массив символов).
-----------------	--

Возвращаемое значение:

Указатель на неизменяемую строку.

Пример вызова функции:

```
new Fixed:message = strfixed("Hello World");  
printf("%s", _:message); // Вывод неизменяемой строки
```

fmul

Умножение чисел с фиксированной запятой.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

Fixed:fmul(Fixed:oper1, Fixed:oper2)

Параметры:

oper1	Первый множитель (число с фиксированной запятой).
oper2	Второй множитель (число с фиксированной запятой).

Возвращаемое значение:

Результат умножения двух чисел с фиксированной запятой.

Пример вызова функции:

```
new Fixed:result = Fixed:fmul(Fixed:1.5, Fixed:2.0); // вернет Fixed:3.0  
new Fixed:price = Fixed:fmul(Fixed:10.99, Fixed:0.85); // расчет скидки
```

fdiv

Деление числа с фиксированной запятой.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

Fixed:fdiv(Fixed:dividend, Fixed:divisor)

Параметры:

dividend	Делимое (число с фиксированной запятой).
divisor	Делитель (число с фиксированной запятой). Не должен быть равен 0.

Возвращаемое значение:

Результат деления числа с фиксированной запятой.

Пример вызова функции:

```
new Fixed:result = Fixed:fdiv(Fixed:3.0, Fixed:2.0); // вернет Fixed:1.5  
new Fixed:ratio = Fixed:fdiv(Fixed:100.0, Fixed:3.0); // вернет Fixed:33.333...
```

ffract

Извлечение дробной части числа с фиксированной запятой.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

Fixed:ffract(Fixed:value)

Параметры:

value	Число с фиксированной запятой для извлечения дробной части.
--------------	---

Возвращаемое значение:

Дробная часть числа в диапазоне [0.0, 1.0) с фиксированной точностью.

Пример вызова функции:

```
new Fixed:num = Fixed:3.14159;  
new Fixed:fraction = Fixed:ffract(num); // вернет Fixed:0.14159  
new Fixed:whole = num - fraction; // целая часть 3.0
```

fround

Округление числа с фиксированной запятой согласно указанному методу.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

```
fround(value, fround_method:method=fround_round)
```

Параметры:

value	Число с фиксированной запятой для округления.
method	Метод округления (по умолчанию fround_round — математическое округление).

Возвращаемое значение:

Округленное значение с фиксированной запятой.

Пример вызова функции:

```
new Fixed:num = Fixed:3.6;  
new Fixed:r1 = fround(num); // 4.0 (по умолчанию математическое округление)
```

fpower

Возведение числа с фиксированной запятой в указанную степень.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

Fixed:fpower(Fixed:value, exponent)

Параметры:

value	Основание степени (число с фиксированной запятой).
exponent	Показатель степени (целое число).

Возвращаемое значение:

Результат возведения в степень с фиксированной запятой.

Пример вызова функции:

```
new Fixed:base = Fixed:2.5;  
new Fixed:result1 = Fixed:fpower(base, 2); // 6.25 (2.5 в квадрате)  
new Fixed:result2 = Fixed:fpower(base, 3); // 15.625 (2.5 в кубе)  
new Fixed:result3 = Fixed:fpower(Fixed:10.0, -1); // 0.1 (10 в степени -1)
```

fsqroot

Вычисление квадратного корня из числа с фиксированной запятой.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

Fixed:fsqroot(Fixed:value)

Параметры:

value	Число с фиксированной запятой для извлечения корня. Не должно быть отрицательным.
--------------	---

Возвращаемое значение:

Квадратный корень числа с фиксированной запятой.

Пример вызова функции:

```
new Fixed:num1 = fixed(25.0);  
new Fixed:sqrt1 = Fixed:fsqroot(num1); // 5.0
```

```
new Fixed:num2 = fixed(2.0);  
new Fixed:sqrt2 = Fixed:fsqroot(num2); // ~1.4142
```

```
new Fixed:num3 = fixed(0.25);  
new Fixed:sqrt3 = Fixed:fsqroot(num3); // 0.5
```

fabs

Получение абсолютного значения числа с фиксированной запятой.

Для использования функции подключите файл:

```
#include <fixed.inc>
```

Формат функции:

Fixed:fabs(Fixed:value)

Параметры:

value	Число с фиксированной запятой для вычисления абсолютного значения.
--------------	--

Возвращаемое значение:

Абсолютное значение (модуль) числа с фиксированной запятой.

Пример вызова функции:

```
new Fixed:num1 = fixed(3.14);  
new Fixed:abs1 = Fixed:fabs(num1); // 3.14  
  
new Fixed:num2 = fixed(-2.71);  
new Fixed:abs2 = Fixed:fabs(num2); // 2.71  
  
new Fixed:num3 = fixed(0.0);  
new Fixed:abs3 = Fixed:fabs(num3); // 0.0
```

Стандартные функции Rown для чисел с плавающей запятой

Список функций	Описание
float	Преобразование числа в формат с плавающей запятой.
strfloat	Преобразование строки в число с плавающей запятой.
floatmul	Умножение чисел с плавающей запятой.
floatdiv	Деление числа с плавающей запятой.
floatadd	Сложение чисел с плавающей запятой.
floatsub	Вычитание чисел с плавающей запятой.
floatfract	Извлечение дробной части числа с плавающей запятой.
floatround	Округление числа с плавающей запятой согласно указанному методу.
floatcmp	Сравнение двух чисел с плавающей запятой.
floatsgroot	Вычисление квадратного корня из числа с плавающей запятой.
floatpower	Возведение числа с плавающей запятой в указанную степень.
floatlog	Вычисление логарифма числа с плавающей запятой по указанному основанию.
floatsin	Вычисление синуса угла для чисел с плавающей запятой.
floatcos	Вычисление косинуса угла для чисел с плавающей запятой.
floattan	Вычисление тангенса угла для чисел с плавающей запятой.
floatabs	Получение абсолютного значения числа с плавающей запятой.

float

Преобразование числа в формат с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:float(value)

Параметры:

value	Число для преобразования (целое, строка или число с фиксированной запятой).
-------	---

Возвращаемое значение:

Значение с плавающей запятой.

Пример вызова функции:

```
new Float:a = Float:float(5); // Из целого числа  
new Float:b = Float:float("3.14"); // Из строки  
new Float:c = Float:float(fixed(2)); // Из числа с фиксированной запятой
```

strfloat

Преобразование строки в число с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:strfloat(const string[])

Параметры:

string[]	Строка для преобразования (должна содержать числовое значение).
-----------------	---

Возвращаемое значение:

Число с плавающей запятой, соответствующее переданной строке.

Пример вызова функции:

```
new Float:val1 = Float:strfloat("3.1415"); // возвращает 3.1415  
new Float:val2 = Float:strfloat("-2.718"); // возвращает -2.718  
new Float:val3 = Float:strfloat("invalid"); // возвращает 0.0
```

floatmul

Умножение чисел с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatmul(Float:oper1, Float:oper2)

Параметры:

oper1	Первый множитель (число с плавающей запятой).
oper2	Второй множитель (число с плавающей запятой).

Возвращаемое значение:

Результат умножения двух чисел с плавающей запятой.

Пример вызова функции:

```
new Float:a = Float:floatmul(3.0, 2.5); // возвращает 7.5  
new Float:b = Float:floatmul(-1.5, 4.0); // возвращает -6.0
```

floatdiv

Деление числа с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatdiv(Float:dividend, Float:divisor)

Параметры:

dividend	Делимое (число с плавающей запятой).
divisor	Делитель (число с плавающей запятой). Не должен быть равен 0.0.

Возвращаемое значение:

Результат деления числа с плавающей запятой. В случае деления на ноль возвращает специальное значение INFINITY (типа Float).

Пример вызова функции:

```
new Float:a = Float:floatdiv(10.0, 2.0); // возвращает 5.0  
new Float:b = Float:floatdiv(1.0, 3.0); // возвращает 0.333...  
new Float:c = Float:floatdiv(5.0, 0.0); // возвращает INFINITY
```

floatadd

Сложение чисел с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatadd(Float:oper1, Float:oper2)

Параметры:

oper1	Первое слагаемое (число с плавающей запятой).
oper2	Второе слагаемое (число с плавающей запятой).

Возвращаемое значение:

Результат сложения чисел с плавающей запятой.

Пример вызова функции:

```
new Float:result = Float:floatadd(2.5, 3.1); // возвращает 5.6  
new Float:sum = Float:floatadd(-1.2, 0.5); // возвращает -0.7
```

floatsub

Вычитание чисел с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatsub(Float:oper1, Float:oper2)

Параметры:

oper1	Уменьшаемые (число с плавающей запятой).
oper2	Вычитаемое (число с плавающей запятой).

Возвращаемое значение:

Результат вычитания.

Пример вызова функции:

```
new Float:result1 = Float:floatsub(5.0, 2.5); // возвращает 2.5  
new Float:result2 = Float:floatsub(1.0, 1.1); // возвращает -0.1  
new Float:result3 = Float:floatsub(INFINITY, INFINITY); // возвращает NaN
```

floatfract

Извлечение дробной части числа с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatfract(Float:value)

Параметры:

value	Число с плавающей запятой для извлечения дробной части.
--------------	---

Возвращаемое значение:

Дробная часть числа в диапазоне [0.0, 1.0) или (-1.0, 0.0] для отрицательных чисел.

Пример вызова функции:

```
new Float:f1 = Float:floatfract(3.1415); // возвращает 0.1415  
new Float:f2 = Float:floatfract(-2.718); // возвращает -0.718  
new Float:f3 = Float:floatfract(5.0); // возвращает 0.0
```

floatround

Округление числа с плавающей запятой согласно указанному методу.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

```
floatround(Float:value, floatround_method:method=floatround_round)
```

Параметры:

value	Число с плавающей запятой для округления.
method	Метод округления (по умолчанию floatround_round — математическое округление).

Возвращаемое значение:

Округленное значение с плавающей запятой.

Пример вызова функции:

```
new Float:a = floatround(3.3); // 3.0 (по умолчанию математическое округление)
```

floatcmp

Сравнение двух чисел с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

floatcmp(Float:oper1, Float:oper2)

Параметры:

oper1	Первое число с плавающей запятой для сравнения.
oper2	Второе число с плавающей запятой для сравнения.

Возвращаемое значение:

- -1, если **oper1** < **oper2**.
- 0, если **oper1** = **oper2**.
- 1, если **oper1** > **oper2**.

Пример вызова функции:

```
new result1 = floatcmp(3.14, 3.14); // 0 (равны)  
new result2 = floatcmp(1.0, 1.0001); // -1  
new result3 = floatcmp(5.0, 4.9); // 1
```

floatsgroot

Вычисление квадратного корня из числа с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatsgroot(Float:value)

Параметры:

value	Число с плавающей запятой для извлечения корня. Не должно быть отрицательным.
--------------	---

Возвращаемое значение:

Квадратный корень числа с плавающей запятой.

Пример вызова функции:

```
new Float:root1 = Float:floatsgroot(25.0); // 5.0  
new Float:root2 = Float:floatsgroot(2.0); // ~1.4142
```

floatpower

Возведение числа с плавающей запятой в указанную степень.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatpower(Float:value, Float:exponent)

Параметры:

value	Основание степени (число с плавающей запятой).
exponent	Показатель степени (число с плавающей запятой).

Возвращаемое значение:

Результат возведения в степень с плавающей запятой.

Пример вызова функции:

```
new Float:res1 = Float:floatpower(2.0, 3.0); // 8.0  
new Float:res2 = Float:floatpower(4.0, 0.5); // 2.0 (квадратный корень)  
new Float:res3 = Float:floatpower(-1.0, 2.0); // 1.0
```

floatlog

Вычисление логарифма числа с плавающей запятой по указанному основанию.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatlog(Float:value, Float:base=10.0)

Параметры:

value	Число с плавающей запятой для логарифмирования, больше 0.
base	Основание логарифма (число с плавающей запятой, больше 0 и не равно 1, по умолчанию 10.0).

Возвращаемое значение:

Значение логарифма с плавающей запятой.

Пример вызова функции:

```
new Float:log1 = Float:floatlog(100.0); // 2.0 (по основанию 10)  
new Float:log2 = Float:floatlog(8.0, 2.0); // 3.0 (логарифм по основанию 2)  
new Float:log3 = Float:floatlog(2.71828); // ~1.0 (натуральный логарифм)
```

floatsin

Вычисление синуса угла для чисел с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatsin(Float:value, anglemode:mode=radian)

Параметры:

value	Угол для вычисления синуса (число с плавающей запятой).
mode	Единицы измерения угла (по умолчанию radian — радианы (0–2π)).

Возвращаемое значение:

Значение синуса с плавающей запятой в диапазоне [–1.0, 1.0].

Пример вызова функции:

```
new Float:sin1 = Float:floatsin(1.5708); // ~1.0 (sin(π/2) радиан)
```

floatcos

Вычисление косинуса угла для чисел с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatcos(Float:value, anglemode:mode=radian)

Параметры:

value	Угол для вычисления косинуса (число с плавающей запятой).
mode	Единицы измерения угла (по умолчанию radian — радианы (0–2π)).

Возвращаемое значение:

Значение косинуса с плавающей запятой в диапазоне [–1.0, 1.0].

Пример вызова функции:

```
new Float:cos1 = Float:floatcos(0.0); // 1.0 (cos(0) радиан)
```

floattan

Вычисление тангенса угла для чисел с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floattan(Float:value, anglemode:mode=radian)

Параметры:

value	Угол для вычисления тангенса (число с плавающей запятой).
mode	Единицы измерения угла (по умолчанию radian — радианы (0–2π)).

Возвращаемое значение:

Значение тангенса с плавающей запятой.

Пример вызова функции:

```
new Float:tan1 = Float:floattan(0.7854); // ~1.0 (tan(π/4) радиан)
```

floatabs

Получение абсолютного значения числа с плавающей запятой.

Для использования функции подключите файл:

```
#include <float.inc>
```

Формат функции:

Float:floatabs(Float:value)

Параметры:

value	Число с плавающей запятой для вычисления абсолютного значения.
-------	--

Возвращаемое значение:

Абсолютное значение (модуль) числа с плавающей запятой.

Пример вызова функции:

```
new Float:abs1 = Float:floatabs(3.14); // 3.14  
new Float:abs2 = Float:floatabs(-2.71); // 2.71  
new Float:abs3 = Float:floatabs(0.0); // 0.0
```

Стандартные функции Pawk для строк

Список функций	Описание
<u>strlen</u>	Получение длины строки.
<u>strpack</u>	Копирование и упаковка строки-источника в строку-назначение.
<u>strunpack</u>	Распаковка упакованной строки в обычный строковый формат.
<u>strcopy</u>	Копирование содержимого одной строки в другую с контролем переполнения.
<u>strcat</u>	Объединение двух строк.
<u>strmid</u>	Копирование части содержимого одной строки в другую.
<u>strins</u>	Вставка подстроки в указанную позицию строки.
<u>strdel</u>	Удаление части строки между указанными позициями.
<u>strcmp</u>	Сравнение двух строк.
<u>strfind</u>	Поиск первого вхождения подстроки в строке.
<u>strval</u>	Преобразование строкового представления числа в целочисленное значение.
<u>valstr</u>	Преобразование целочисленного значения в строковое представление.
<u>ispacked</u>	Проверка вида строки (упакована или не упакована).
<u>strformat</u>	Форматирование строки согласно заданному шаблону.
<u>uudecode</u>	Декодирование строки из формата UUencode в бинарные данные.
<u>uuencode</u>	Кодирование бинарных данных в формат UUencode.
<u>memcpy</u>	Копирование данных из одного массива в другой.

strlen

Получение длины строки.

Для использования функции подключите файл:

```
#include <string.h>
```

Формат функции:

```
strlen(const string[])
```

Параметры:

string[]	Строка, длину которой требуется узнать.
-----------------	---

Возвращаемое значение:

Длина строки (целое число).

Примечание. Данную функцию не рекомендуется использовать на пустой строке: если строка не пустая, функция обойдет все символы в строке, чтобы найти символ конца строки «\0» и на основе его позиции узнать длину. Чем длиннее строка, тем больше времени будет затрачено на обход всех символов.

Пример вызова функции:

```
new len = strlen("Пример строки");  
// Вывод: "Длина строки: 13 символов"  
printf("Длина строки: %d символов", len);
```

strpack

Копирование и упаковка строки-источника в строку-назначение.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

```
strpack(dest[], const source[], maxlength=sizeof dest)
```

Параметры:

dest[]	Массив для записи упакованной строки.
source[]	Строка, которую требуется упаковать. Если строка уже упакована, то функция копирует ее.
maxlength	Максимальная длина упакованной строки (по умолчанию размер массива dest[]). Необязательный параметр.

Возвращаемое значение:

Длина упакованной строки.

Пример вызова функции:

```
// Базовое использование  
new packed[32];  
strpack(packed, "Hello World");
```

```
// С указанием максимальной длины  
new small[10];  
strpack(small, "Very long string", sizeof small);
```

strunpack

Распаковка упакованной строки в обычный строковый формат.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

```
strunpack(dest[], const source[], maxlenh=sizeof dest)
```

Параметры:

dest[]	Массив для записи распакованной строки.
source[]	Строка, которую требуется распаковать. Если строка уже распакована, то функция копирует ее.
maxlenh	Максимальная длина распакованной строки (по умолчанию размер массива dest[]). Необязательный параметр.

Возвращаемое значение:

Длина распакованной строки.

Пример вызова функции:

```
// Распаковка упакованной строки
new packed[32], unpacked[32];
strpack(packed, "Compressed text");
strunpack(unpacked, packed);

// С ограничением длины вывода
new short[10];
strunpack(short, packed, sizeof short);
```

strcpy

Копирование содержимого одной строки в другую с контролем переполнения.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

```
strcpy(dest[], const source[], maxlenh=sizeof dest)
```

Параметры:

dest[]	Массив для копирования содержимого строки.
source[]	Строка, содержимое которой требуется скопировать.
maxlength	Максимальное количество копируемых символов (по умолчанию размер массива dest[]). Необязательный параметр.

Возвращаемое значение:

Количество скопированных символов (целое число).

Пример вызова функции:

```
// Базовое использование  
new buffer[32];  
strcpy(buffer, "Hello World");
```

```
// Копирование с ограничением длины  
new small[10];  
strcpy(small, "Very long string", sizeof small);
```

strcat

Объединение двух строк.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

strcat(dest[], const source[], maxlenh=sizeof dest)

Параметры:

dest[]	Строка, в конец которой требуется добавить содержимое строки source[] .
source[]	Строка, содержимое которой требуется добавить к строке dest[] .
maxlenh	Максимальное количество копируемых символов (по умолчанию размер массива dest[]). Необязательный параметр.

Возвращаемое значение:

Количество символов, добавленных в строку **dest[]** (целое число).

Примечание. Строка **dest[]** после объединения сохраняет исходный упакованный/неупакованный вид. Например, если перед вызовом функции строка была упакована, то она и после объединения останется упакованной.

Пример вызова функции:

```
// Базовое объединение строк  
new text[32] = "Hello";  
strcat(text, " World");  
printf("%s", text); // "Hello World"
```

```
// Безопасное объединение с контролем длины  
new buffer[10] = "Test";  
strcat(buffer, "12345", sizeof buffer);  
printf("%s", buffer); // "Test1234" (обрезано до размера буфера)
```

strmid

Копирование части содержимого одной строки в другую.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

strmid(dest[], const source[], start=0, end=cellmax, maxlength=sizeof dest)

Параметры:

dest[]	Строка, в которую требуется скопировать содержимое строки source[] .
source[]	Строка, часть содержимого которой требуется скопировать в строку dest[] .
start	Позиция начала копирования (нумерация начинается с 0).
end	Позиция завершения копирования. Копируются символы из диапазона от start до (end – 1).
maxlength	Максимальное количество копируемых символов (по умолчанию размер массива dest[]). Необязательный параметр.

Возвращаемое значение:

Количество скопированных символов (целое число).

Примечание. Вид строки **dest[]** (упакована или не упакована) зависит от вида строки **source[]**.

Пример вызова функции:

```
new result[32];
strmid(result, "Hello World", 6, 11);
printf("%s", result); // "World"
```

strins

Вставка подстроки в указанную позицию строки.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

bool: strins(string[], const substr[], index, maxlenh=sizeof string)

Параметры:

string[]	Строка, в которую требуется вставить подстроку substr[] .
substr[]	Подстрока для вставки в строку string[] .
index	Позиция начала вставки (0 — начало строки).
maxlength	Максимальный размер массива string[] . Необязательный параметр.

Возвращаемое значение:

true — вставка выполнена успешно, false — вставка невозможна.

Примечание. Строка **string[]** сохраняет исходный упакованный/неупакованный вид после вставки подстроки. Например, если перед вызовом функции строка была упакована, то и после вставки она останется упакованной.

Пример вызова функции:

```
// Вставка в середину строки
new text[32] = "Hello World";
bool: strins(text, "Beautiful ", 6);
printf("%s", text); // "Hello Beautiful World"

// Вставка с проверкой границ
new buffer[20] = "OpenAI";
bool: strins(buffer, "ChatGPT ", 0, sizeof buffer);
printf("%s", buffer); // "ChatGPT OpenAI" (если хватает места)

// Вставка в конец строки (аналогично strcat)
new str[32] = "Start";
bool: strins(str, "End", strlen(str));
printf("%s", str); // "StartEnd"
```

strdel

Удаление части строки между указанными позициями.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

bool: strdel(string[], start, end)

Параметры:

string[]	Строка, из которой требуется удалить символы.
start	Позиция начала удаления (0 — начало строки).
end	Позиция завершения удаления. Удаляются символы из диапазона от start до (end - 1).

Возвращаемое значение:

true — удаление выполнено успешно, false — удаление невозможно.

Пример вызова функции:

```
// Удаление части строки  
new text[32] = "Hello Beautiful World";  
bool: strdel(text, 6, 15);  
printf("%s", text); // "Hello World"
```

```
// Удаление с начала строки  
new str[32] = "PrefixData";  
bool: strdel(str, 0, 6);  
printf("%s", str); // "Data"
```

```
// Удаление до конца строки  
new buffer[32] = "MainTextExtra";  
bool: strdel(buffer, 8, strlen(buffer));  
printf("%s", buffer); // "MainText"
```

strcmp

Сравнение двух строк.

Для использования функции подключите файл:

```
#include <string.h>
```

Формат функции:

```
strcmp(const string1[], const string2[], bool:ignorecase=false, length=cellmax)
```

Параметры:

string1[]	Первая строка для сравнения.
string2[]	Вторая строка для сравнения.
ignorecase	Флаг регистронезависимого сравнения (по умолчанию false — учитывать регистр). Необязательный параметр.
length	Максимальное количество символов для сравнения. Необязательный параметр.

Возвращаемое значение:

- 0, если строки равны.
- 1, если на *i*-й позиции символы различаются и **string1[i] > string2[i]**.
- -1, если на *i*-й позиции символы различаются и **string1[i] < string2[i]**.

Пример вызова функции:

```
result = strcmp("1234", "1234");  
printf("result: %d", result); // "result: 0"  
  
result = strcmp("1234", "5678");  
printf("result: %d", result); // "result: -1" ('1' < '5')  
  
result = strcmp("efgh", "abcd");  
printf("result: %d", result); // "result: 1" ('e' > 'a')
```

strfind

Поиск первого вхождения подстроки в строке.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

```
strfind(const string[], const sub[], bool:ignorecase=false, index=0)
```

Параметры:

string[]	Строка, в которой требуется найти подстроку sub[] .
sub[]	Подстрока, которую требуется найти в строке string[] .
ignorecase	Флаг регистронезависимого поиска (по умолчанию false — учитывать регистр). Необязательный параметр.
index	Позиция начала поиска (по умолчанию 0). Необязательный параметр.

Возвращаемое значение:

Позиция первого вхождения подстроки **sub[]** в строку **string[]** (целое число). И -1, если подстрока не найдена.

Пример вызова функции:

```
// Простой поиск
new pos = strfind("Hello World", "World");
// pos == 6

// Регистронезависимый поиск
pos = strfind("Sample Text", "text", true);
// pos == 7

// Поиск с указанием начальной позиции
pos = strfind("ababab", "ab", false, 2);
// pos == 2

// Проверка наличия подстроки
if (strfind("main string", "sub") != -1) {
    print("Подстрока найдена");
}
```

strval

Преобразование строкового представления числа в целочисленное значение.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

```
strval(const string[], index=0)
```

Параметры:

string[]	Строка, которую требуется преобразовать в число.
index	Позиция в строке, с которой требуется начать разбор (по умолчанию 0). Необязательный параметр.

Возвращаемое значение:

Целочисленное значение. Если преобразование не выполнено, то возвращает 0.

Пример вызова функции:

```
new value = strval("42", 0);  
// value == 42
```

valstr

Преобразование целочисленного значения в строковое представление.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

valstr(dest[], value, bool:pack=true)

Параметры:

dest[]	Массив для записи строкового представления числа.
value	Целочисленное значение для преобразования.
pack	Флаг упаковки строки (по умолчанию true — сохранить как упакованную строку). Необязательный параметр.

Возвращаемое значение:

Количество записанных символов (целое число).

Пример вызова функции:

```
new str[10];  
valstr(str, 42);  
printf("%s", str); // "42"
```

ispacked

Проверка вида строки (упакована или не упакована).

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

bool: ispacked(const string[])

Параметры:

string[]	Строка, которую требуется проверить.
-----------------	--------------------------------------

Возвращаемое значение:

true — строка упакована, false — строка не упакована.

Пример вызова функции:

```
new normalStr[] = "Hello";  
if (!ispacked(normalStr)) {  
    print("Строка не упакована");  
}
```

strformat

Форматирование строки согласно заданному шаблону.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

```
strformat(dest[], size=sizeof dest, bool:pack=true, const format[], {Fixed,Float,_}:...)
```

Параметры:

dest[]	Массив для записи отформатированной строки.
size	Максимальная длина отформатированной строки (по умолчанию размер массива dest[]). Необязательный параметр.
pack	Флаг упаковки строки (по умолчанию true — сохранить как упакованную строку). Необязательный параметр.
format[]	Строка формата (шаблон).
...	Переменные аргументы для подстановки в шаблон.

Возвращаемое значение:

Количество записанных символов (целое число).

Пример вызова функции:

```
// Базовое форматирование
new buffer[128];
strformat(buffer, sizeof(buffer), true, "Player: %s, Score: %d", "John", 100);

// Форматирование чисел
new float:value = 3.14159;
strformat(buffer, sizeof(buffer), false, "Value: %.2f", value);

// Создание упакованной строки
new packedStr[64 char];
strformat(packedStr, sizeof(packedStr), true, "ID:%04d", 42);
```

uudecode

Декодирование строки из формата UUencode в бинарные данные.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

uudecode(dest[], const source[], maxlength=sizeof dest)

Параметры:

dest[]	Массив для записи декодированных данных.
source[]	Строка в формате UUencode, которую требуется декодировать.
maxlength	Максимальный размер массива dest[] . Необязательный параметр.

Возвращаемое значение:

Количество декодированных байтов (целое число).

Пример вызова функции:

```
new data[128];
new decoded = uudecode(data, "begin 644 file.txt\nMOF%R96T@<FEP\n`nend");
if (decoded > 0) {
    printf("Декодировано %d байт", decoded);
}
```

uuencode

Кодирование бинарных данных в формат UUencode.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

uuencode(dest[], const source[], numbytes, maxlength=sizeof dest)

Параметры:

dest[]	Массив для записи закодированной строки.
source[]	Бинарные данные для кодирования.
numbytes	Количество байтов для кодирования.
maxlength	Максимальный размер массива dest[] . Необязательный параметр.

Возвращаемое значение:

Количество записанных символов (целое число).

Примечание. В конце строки с закодированными данными добавляется символ «\n».

Пример вызова функции:

```
// Кодирование бинарных данных
new data[10] = {0x54, 0x45, 0x53, 0x54, 0x00};
new encoded[128];
new result = uuencode(encoded, data, 5, sizeof(encoded));
if (result > 0) {
    printf("Закодированная строка: %s", encoded);
}
```

memcpy

Копирование данных из одного массива в другой.

Для использования функции подключите файл:

```
#include <string.inc>
```

Формат функции:

memcpy(dest[], const source[], index=0, numbytes, maxlength=sizeof dest)

Параметры:

dest[]	Массив, в который требуется скопировать данные.
source[]	Массив, из которого требуется скопировать данные.
index	Номер байта в массиве source[] , с которого требуется начать копирование (по умолчанию 0). Необязательный параметр.
numbytes	Количество байтов для кодирования.
maxlength	Максимальный размер массива dest[] . Необязательный параметр.

Возвращаемое значение:

1 — копирование выполнено, 0 — копирование невозможно (указаны неправильные параметры).

Пример вызова функции:

```
// Копирование всего массива
new src[10] = {1, 2, 3, 4, 5};
new dest[10];
memcpy(dest, src, 0, sizeof(src));

// Частичное копирование с указанием размера буфера
new buffer[5];
memcpy(buffer, src, 2, 3 * 4, sizeof(buffer)); // Копирует 3 элемента (по 4 байта) начиная с индекса 2
```

Стандартные функции Rawp для времени и таймера

Список функций	Описание
<u>gettime</u>	Получение текущего времени с разбивкой на часы, минуты, секунды.
<u>settime</u>	Установка системного времени.
<u>getdate</u>	Получение текущей даты.
<u>setdate</u>	Установка даты.
<u>settimestamp</u>	Установка системного времени по Unix-временной метке.
<u>cvttimestamp</u>	Конвертация Unix-временной метки в компоненты даты и времени.
<u>settimer</u>	Создание и запуск таймера с заданным интервалом.
<u>tickcount</u>	Получение количества миллисекунд, прошедших с момента запуска системы.
<u>delay</u>	Приостановка выполнения текущего потока на указанное время.

gettime

Получение текущего времени с разбивкой на часы, минуты, секунды.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

```
gettime(&hour=0, &minute=0, &second=0)
```

Параметры:

&hour	Ссылка на переменную для получения часа (0–23). Необязательный параметр.
&minute	Ссылка на переменную для получения минут (0–59). Необязательный параметр.
&second	Ссылка на переменную для получения секунд (0–59). Необязательный параметр.

Возвращаемое значение:

Текущее Unix-время (`time_t`) в секундах. Часовая, минутная и секундная составляющие возвращаются по отдельности через параметры **hour**, **minute** и **second** соответственно.

Примечание. Возвращаемое функцией значение (Unix-время), а также значения, возвращаемые через параметры **hour**, **minute** и **second**, обозначают время в часовом поясе UTC+0 (GMT+0).

Пример вызова функции:

```
new h, m, s;  
new currentTime = gettime(h, m, s);  
printf("Время: %02d:%02d:%02d (%d mc)", h, m, s, currentTime);
```

settime

Установка системного времени.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

```
settime(hour=cellmin, minute=cellmin, second=cellmin)
```

Параметры:

hour	Часовая составляющая для установки (0–23). Необязательный параметр, при пропуске текущий час не изменяется.
minute	Минутная составляющая для установки (0–59). Необязательный параметр, при пропуске текущая минута не изменяется.
second	Секундная составляющая для установки (0–59). Необязательный параметр, при пропуске текущая секунда не изменяется.

Возвращаемое значение:

0.

Пример вызова функции:

```
// Установка полного времени  
settime(14, 30, 0); // Устанавливает 14:30:00
```

```
// Установка только часов  
settime(12); // Устанавливает 12:XX:XX (минуты и секунды не изменяются)
```

getdate

Получение текущей даты.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

```
getdate(&year=0, &month=0, &day=0)
```

Параметры:

&year	Ссылка на переменную для получения года (4 цифры). Необязательный параметр.
&month	Ссылка на переменную для получения месяца (1–12). Необязательный параметр.
&day	Ссылка на переменную для получения дня месяца (1–31). Необязательный параметр.

Возвращаемое значение:

Номер дня с начала года (отсчет начинается с 1, т. е. 1 января считается 1-м днем). Год, месяц и день возвращаются по отдельности через параметры **year**, **month** и **day** соответственно.

Примечание. Все возвращаемые функцией значения (как напрямую, так и через параметры **year**, **month** и **day**) основываются на часовом поясе UTC+0 (GMT+0).

Пример вызова функции:

```
// Получение полной даты
new y, m, d;
new ms = getdate(y, m, d);
printf("Дата: %04d-%02d-%02d, время с полуночи: %d мс", y, m, d, ms);

// Получение только года
new year;
getdate(year);
printf("Текущий год: %d", year);
```

setdate

Установка даты.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

```
setdate(year=cellmin, month=cellmin, day=cellmin)
```

Параметры:

year	Год для установки (4 цифры). Необязательный параметр, при пропуске текущий год не изменяется.
month	Месяц для установки (1–12). Необязательный параметр, при пропуске текущий месяц не изменяется.
day	День для установки (1–31). Необязательный параметр, при пропуске текущий день не изменяется.

Возвращаемое значение:

0.

Пример вызова функции:

```
// Установка полной даты  
setdate(2023, 12, 31); // Устанавливает 31 декабря 2023 года
```

```
// Установка только года  
setdate(2024); // Устанавливает 2024 год, месяц и день не изменяются
```

settimestamp

Установка системного времени по Unix-временной метке.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

```
settimestamp(seconds1970)
```

Параметры:

seconds1970	Количество секунд, прошедших с 1 января 1970 года.
--------------------	--

Возвращаемое значение:

Нет.

Пример вызова функции:

```
// Установка конкретной даты и времени (1 января 2023 00:00:00 UTC)  
settimestamp(1672531200);
```

cvttimestamp

Конвертация Unix-временной метки в компоненты даты и времени.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

```
cvttimestamp(seconds1970, &year=0, &month=0, &day=0, &hour=0, &minute=0, &second=0)
```

Параметры:

seconds1970	Количество секунд, прошедших с 1 января 1970 года.
&year	Ссылка на переменную для получения года (4 цифры). Необязательный параметр.
&month	Ссылка на переменную для получения месяца (1–12). Необязательный параметр.
&day	Ссылка на переменную для получения дня месяца (1–31). Необязательный параметр.
&hour	Ссылка на переменную для получения часа (0–23). Необязательный параметр.
&minute	Ссылка на переменную для получения минут (0–59). Необязательный параметр.
&second	Ссылка на переменную для получения секунд (0–59). Необязательный параметр.

Возвращаемое значение:

Нет.

Пример вызова функции:

```
// Полное преобразование метки времени
new y, m, d, h, mn, s;
cvttimestamp(1672531200, y, m, d, h, mn, s); // 1 января 2023 00:00:00
printf("Дата: %04d-%02d-%02d %02d:%02d:%02d", y, m, d, h, mn, s);

// Получение только даты
new year, month, day;
cvttimestamp(1672531200, year, month, day);
printf("Дата: %04d-%02d-%02d", year, month, day);

// Получение только времени
new hour, minute;
cvttimestamp(1672531200, .hour = hour, .minute = minute);
printf("Время: %02d:%02d", hour, minute);
```

settimer

Создание и запуск таймера с заданным интервалом.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

```
settimer(milliseconds, bool: singleshot=false)
```

Параметры:

milliseconds	Интервал срабатывания таймера в миллисекундах (1–2147483647).
singleshot	Режим работы таймера (по умолчанию false — периодический).

Возвращаемое значение:

ID созданного таймера.

Пример вызова функции:

```
// Создание периодического таймера (срабатывает каждую секунду)  
new Timer:periodic_timer = settimer(1000);
```

tickcount

Получение количества миллисекунд, прошедших с момента запуска системы.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

```
tickcount(&granularity=0)
```

Параметры:

&granularity	Ссылка на переменную для получения точности таймера, в миллисекундах. Необязательный параметр.
-------------------------	---

Возвращаемое значение:

Количество миллисекунд с момента запуска системы.

Пример вызова функции:

```
printf("С момента запуска системы прошло %d мс", tickcount());
```

delay

Приостановка выполнения текущего потока на указанное время.

Для использования функции подключите файл:

```
#include <time.inc>
```

Формат функции:

delay(milliseconds)

Параметры:

milliseconds	Время задержки в миллисекундах (0–2147483647).
---------------------	--

Возвращаемое значение:

Нет.

Пример вызова функции:

```
// Простая задержка на 1 секунду  
delay(1000);
```



ООО НПО «ТехноКом»

www.glonassgps.ru

02.03.2026

info@tk-nav.ru