МАРАФОН
CAN технология

CANopen библиотека

Руководство программиста

Код проекта: **0001**$_h$

Москва, 2017

Оглавление

Введение

Библиотека CANopen позволяет разрабатывать программное обеспечение slave и master устройств, совместимых со спецификациями CiA 301 v. 4.2 и EN50325-5. Библиотека поддерживает LSS slave устройства согласно CiA DSP 305 v. 2.2. Программное обеспечение библиотеки написано на языке ANSI C с учетом масштабируемости и переносимости на различные платформы.

Для доступа к сети на канальном уровне библиотека использует API драйвера CHAI. Зависимости кода библиотеки от среды исполнения выделены в отдельный модуль. Таким образом, исходный код библиотеки не зависит от конкретной платформы и одинаков как для приложений, встраиваемых в микроконтроллеры, так и для задач, работающих под управлением операционных систем общего назначения: Windows, Linux и других.

Основные характеристики библиотеки

- Библиотека обеспечивает работу приложений в режиме жесткого реального времени. Ее архитектура основана на повторно–входимых компонентах, которые допускают асинхронное обращение к ним со стороны прикладной программы.
- Раздел объектного словаря для коммуникаций реализует полное реконфигурирование в соответствии со стандартами CiA 301 и EN50325-5.
- Инициализация всех коммуникационных объектов производится согласно предопределенному распределению CAN идентификаторов.

Функциональность библиотеки

- CANopen SDO протокол поддерживается во всех предусмотренных стандартом режимах: ускоренном, сегментированном и блочном.
- Реализованы все виды PDO протоколов (cyclic, acyclic, synchronous, asynchronous, RTR only). Может использоваться как статическое, так и динамическое PDO отображение.
- Протокол синхронизации SYNC обеспечивает работу как с SYNC счетчиком, так и без его использования.
- Поддерживаются все протоколы сетевого менеджера (NMT).
- Реализованы протоколы контроля ошибок: сердцебиения (Heartbeat) и охраны узла (Node Guarding).
- Поддерживается протокол начальной загрузки Boot-up.
- Реализовано полное семейство LSS протоколов, включая Fastscan.
- Компоненты библиотеки поддерживают стандарт EN50325-5: функционально безопасные коммуникации на основе CANopen.

Ограничения библиотеки

- Максимальный размер любого объекта не должен превышать $7FFFFFFF_h$ (2147483647) байт.
- Минимальное значение периода CANopen таймера составляет 100 микросекунд (частота не более 10 Кгц).
- Действующие версии стандарта CANopen поддерживают работу только с 11-битовыми идентификаторами. 29-битовые идентификаторы являются зарезервированными и не используются в протоколе CANopen. Библиотека игнорирует все входящие кадры с 29-битовыми идентификаторами.

Оптимизация кода библиотеки

При использовании библиотеки настоятельно рекомендуется отключать алгоритмическую оптимизацию исходного кода компилятором среды разработки. Применение оптимизации нередко нарушает соответствие алгоритмов, написанных на языке высокого уровня и машинного кода, генерируемого компилятором. Использование методик частичного подавления оптимизации, например, дополнительное объявление переменных «изменяемыми» (volatile) не дает гарантии того, что все вносимые алгоритмические ошибки и побочные эффекты оптимизации будут устранены.

Соглашения по документации

Библиотека разработана на основе стандартов международных организаций CAN in Automation и CENELEC.

CiA 301	v. 4.2	Спецификация прикладного уровня и коммуникационного профиля CAN, определяющая функциональность CANopen устройств.
EN50325-5	2010	Функционально безопасные коммуникации на основе CANopen.
DSP 305	v. 2.2	Службы установки сетевого уровня LSS.
CiA 303 ч. 3	v. 1.4	Проектные рекомендации по использованию светодиодов.
CiA 306	v. 1.3	Определяет формат и содержимое электронных спецификаций (EDS, DCF), применяемых в конфигурационном инструментарии.

Эталонной технологической силой обладают исключительно оригинальные версии стандартов, которые составлены на английском языке: © CAN in Automation (CiA) e. V.; © CENELEC. Переводы стандартов носят справочно–рекомендательный характер.

Дополнением к данному руководству являются описания:
«Адаптированный slave для ОС Windows»;
«Адаптированный master для ОС Windows»;
«DLL master для ОС Windows с приложением для LabVIEW».

Принятые сокращения

CiA	Международная организация CAN in Automation – "CAN в автоматизации".
LSS	CANopen службы (сервисы) установки уровня. Служат для конфигурирования номера CAN узла и битовой скорости CAN сети.
CAN–ID	Идентификатор CAN кадра канального уровня.
COB–ID	Идентификатор коммуникационного объекта CANopen.
NMT	Сетевой менеджер: определяет объекты управления CANopen сетью.
PDO	Объект данных процесса; обеспечивает обмен компактными данными (до 8 байт) в режиме жесткого реального времени.
RTR	Удаленный запрос объекта.
SDO	Сервисный объект данных; обеспечивает обмен большими объемами данных в режиме мягкого реального времени.
LSB	Наименее значимый (младший) бит или байт.
MSB	Наиболее значимый (старший) бит или байт.
RO	Доступ только по чтению.
WO	Доступ только по записи.
RW	Доступ по чтению и записи.
RWR	Доступ по чтению и записи, асинхронный доступ по чтению (для PDO и SRDO).
RWW	Доступ по чтению и записи, асинхронный доступ по записи (для PDO и SRDO).

GFC	Широковещательная команда прекращения безопасного протокола и перевода устройства в безопасное состояние.
SCL	Безопасный коммуникационный уровень.
SCT	Длительность цикла безопасности.
SRD	Устройство с поддержкой безопасности.
SRDO	Безопасный объект данных.
SRVT	Время достоверности безопасного объекта данных.

Для подробного ознакомления с терминологией рекомендуется использовать CAN словарь, изданный на русском языке организацией CAN in Automation. Электронная версия словаря размещена здесь.

Обозначения основных типов данных

boolean	Логическое значение true/false.
int8	Целое 8 бит со знаком.
unsigned8	Без-знаковое целое 8 бит.
int16	Целое 16 бит со знаком.
unsigned16	Без-знаковое целое 16 бит.
int32	Целое 32 бита со знаком.
unsigned32	Без-знаковое целое 32 бита.
int64	Целое 64 бита со знаком.
unsigned64	Без-знаковое целое 64 бита.
real32	32–х разрядное с плавающей точкой.
real64	64–х разрядное с плавающей точкой.
vis-string	Строка видимых ASCII символов (коды 0 и 20_h..$7E_h$).
octet-string	Байтовая строка (коды 0..255).

Изменения в версиях

Версия 1.2

Добавлена поддержка объекта 1029_h, определяющего поведение CAN устройства при возникновении серьезных ошибок (Error behaviour object). Введена возможность приема CAN кадров как по сигналу или аппаратному прерыванию контроллера, так и в режиме опроса. Произведена оптимизация некоторых алгоритмов работы с CAN сетью и объектным словарем.

Версии 1.2.10 и выше.

В состав библиотеки включен модуль байт–ориентированного динамического PDO отображения, когда в одном PDO может содержаться до восьми объектов, размер каждого из которых должен быть кратным восьми бит. Выбор модуля динамического PDO отображения осуществляется с помощью параметра сборки приложения CAN_DYNAMIC_MAPPING_GRANULARITY.
Внесены изменения, обеспечивающие соответствие библиотеки версии 4.02 стандарта CiA 301.

Версия 1.3

Добавлен двухуровневый аппаратный фильтр входящих кадров CAN контроллера. Может использоваться только при поддержке со стороны драйвера CHAI. Двухуровневый фильтр дает возможность отбирать лишь кадры, предназначенные данному узлу, при условии использования предопределенного распределения CANopen идентификаторов.

Версия 1.4

Добавлен модуль светодиодной индикации состояния CAN устройства в соответствии с «проектными рекомендациями по использованию светодиодов» (CiA 303 часть 3 v. 1.3).
Внесены изменения в модуль объекта синхронизации для контроля потребителем SYNC своевременного получения объекта синхронизации. Внесены изменения в модуль обработки принимаемых PDO для контроля таймаута RPDO до истечения таймера события.
Добавлена функция, облегчающая работу приложения с несколькими коммуникационными интерфейсами при отключенной шине CAN. См. новый раздел «API функций общего управления».
Введены новые регистраторы событий (раздел «API функций, редактируемых пользователем»):

- потребителем не получен объект синхронизации,
- не получено RPDO до истечения таймера события,
- произошло наложение тиков CANopen таймера,
- произошло переполнение выходного CANopen кэша.

Версия 1.5

Полностью реализован объект 1007_h — длительность окна синхронизации.
Полностью реализованы алгоритмы сохранения/восстановления параметров в энергонезависимой памяти (объекты 1010_h, 1011_h).

Версия 1.6

Документация библиотеки переведена в формат pdf.
Введены новые регистраторы событий (раздел «API функций, редактируемых

пользователем»):
- node guarding event со статусом resolved,
- heartbeat event со статусом resolved,
- life guarding event со статусом resolved,
- регистрация NMT состояния узла.

Внесены изменения в модуль объекта ошибок can_obj_errors.c (версии модуля 1.6.1 и выше).

Версия 1.7

Библиотека приведена в соответствие с версией 4.2 стандарта CiA DSP 301 от 07 декабря 2007 г. Основные дополнения связаны с введением SYNC счетчика и двух типов SYNC кадров: с длиной данных 0 и 1 байт. Реализован объект 1019_h (значение переполнения для SYNC счетчика) и субиндекс 6 в TPDO (начальное значение SYNC счетчика). Изменен API ряда функций библиотеки (в качестве параметра дополнительно передается текущее значение SYNC счетчика).

Версия 2.0

В библиотеку добавлена поддержка LSS slave функциональности на основе спецификации CiA DSP 305 v. 2.2 (модуль can_lss_slave.c). Внесены изменения в модуль объекта сохранения параметров в энергонезависимой памяти can_obj_re_store.c для возможности сохранения номера CAN узла и индекса битовой скорости CAN сети. Значение битовой скорости задается ее индексом.

Версия 2.1

В состав библиотеки включена адаптированная slave версия для ОС Windows.
В «API функций slave для приложений, которые взаимодействуют с CANopen» добавлена функция побитовой очистки регистра ошибок (объект 1001_h).
В «API системно–зависимых функций» добавлены функции разрешения работы и блокировки передающего CAN трансивера.
Изменен API функции обработки переполнения CANopen кэша (раздел «API функций, редактируемых пользователем»). В качестве параметра дополнительно передается NMT состояния CAN узла.

Версия 2.2

Введен контроль длины данных входящих CAN кадров для всех принимаемых CANopen объектов. Если длина данных не соответствует требуемой, кадр игнорируется. В предшествующих версиях библиотеки анализ длины кадра производился в соответствии с рекомендациями CiA 301 только для PDO и SYNC объектов.
Введена проверка и запрет идентификаторов ограниченного использования в конфигурируемых COB-ID.
Изменен API функций сохранения номера CAN узла и индекса битовой скорости CAN сети (раздел «API функций slave для приложений, которые взаимодействуют с CANopen»).
Изменен API функции не получения RPDO до истечения его таймера события (раздел «API функций, редактируемых пользователем»). В качестве параметра дополнительно передается индекс коммуникационного объекта не полученного RPDO.
В состав библиотеки включена адаптированная master версия для ОС Windows. Для адаптированных версий составлено отдельное руководство.

Версия 2.3

В состав библиотеки включен CANopen мастер для ОС Windows, реализованный в виде DLL

модуля. В качестве одного из приложений мастера используется модуль сопряжения с пакетом LabVIEW. При этом поддержка DLL версии самой CANopen библиотеки прекращена.
Изменен API функции can_init_system_timer(...) (раздел «API системно–зависимых функций»).
В именах модулей, функций, констант и переменных разделены категории идентификаторов CAN–ID и COB–ID.
Библиотека адаптирована для прохождения Теста соответствия (CANopen conformance test) третьей главной версии.
Состояние PDO (действительно / не действительно) поддерживается вне зависимости от NMT состояния CAN узла.

Версия 3.0

Реализовано расширение CANopen с поддержкой режимов безопасности на основе стандарта EN50325-5 (функционально безопасные коммуникации). Программный код EN50325-5 включен в состав адаптированных версий библиотеки.
Для slave устройств возможна работа по нескольким CAN шинам (до восьми) в режиме ”холодного” резервирования.
Описание реализации безопасного протокола и системы резервирования CAN сетей приведены в документах CANopen_slave.pdf и IOremote.pdf.

Управление версиями модулей

Библиотека поддерживает простейшую систему управления версиями на основе директив C препроцессора. Каждый модуль библиотеки заключен в условный макрос вида:

```
#if CHECK_VERSION(2, 3, 0)
    код модуля библиотеки
#endif
```

Первый аргумент макроса означает главную версию библиотеки, второй – подверсию, третий – номер выпуска. Все модули библиотеки должны иметь одинаковый номер версии и подверсии, а их номер выпуска должен быть не ниже минимального (обычно ноль). При изменении номера версии и подверсии библиотеки используются следующие соглашения:

- если код модуля не изменяется, ему присваивается нулевой номер выпуска;
- если при смене версии или подверсии в код модуля вносятся изменения, ему присваивается первый номер выпуска.

История версий модуля библиотеки сохраняется путем фиксации последнего номера выпуска каждой версии и подверсии в виде комментария C. Например, набор макросов версий модуля

```
#if CHECK_VERSION(2, 2, 0)
//  CHECK_VERSION(2, 1, 0)
//  CHECK_VERSION(2, 0, 0)
//  CHECK_VERSION(1, 7, 1)
```

означает, что текущая версия 2.2.0 идентична версии 1.7.1 данного модуля.

Сборка и установка библиотеки

Приведено описание сборки (компиляции) и установки библиотеки для операционных систем общего назначения (Linux и Windows).

Установ драйвера канального уровня

Установить драйвер канального уровня CAN сети CHAI, руководствуясь инструкциями, размещенными на сайте.

Замечание. Сборка библиотеки в тестовом режиме не требует наличия CAN контроллера и драйвера CHAI.

Структура библиотеки

В директории CANopen содержится поддиректория вида 3.0.x, определяющая номер версии библиотеки. Первое число означает главную версию, второе – подверсию. Все модули библиотеки должны иметь одинаковый номер версии и подверсии, а их номер выпуска должен быть не ниже минимального (обычно ноль). В указанной поддиректории, в свою очередь, размещены следующие три директории:

- Linux – содержит модули для работы в ОС Linux.
- src – «корневая» директория CANopen с исходными кодами библиотеки. Размещение всех модулей приводится относительно этой директории.
- win – сюда записываются файлы проектов (*.sln, *.suo, *.vcxproj, *.vcxproj.*) для среды разработки Microsoft Visual C++ 2010. Эти проекты могут использоваться для сборки конечного приложения на основе библиотеки CANopen.

Операционная система Windows

Для сборки конечного приложения в заголовочном файле \include__can_defines.h следует выбрать тип операционной системы Windows: #define CAN_OS_WIN32 и установить параметр режима сборки конечного приложения CAN_APPLICATION_MODE. При необходимости можно переопределить другие конфигурационные параметры, см. «Параметры режимов и сборки CANopen приложения».

Для компиляции приложения посредством Microsoft Visual C++ 2010 необходимо выполнить следующие операции:

- Задать директории, в которых размещаются заголовочные файлы библиотеки и CHAI драйвера. Например, ..\src\include для файлов CANopen библиотеки и C:\Program Files (x86)\CHAI – 2.10.4\include для заголовочных файлов CHAI драйвера. Навигация: Project –> Properties –> Configuration properties –> C/C++ –> Additional Include Directories.
- Задать директорию, в которой размещается lib файл CHAI драйвера. Например, C:\Program Files (x86)\CHAI – 2.10.4\lib. Навигация: Project –> Properties –> Configuration properties –> Linker –> Additional Library Directories.
- Собрать конечное приложение. Навигация: Build –> Build Solution.

Операционная система Linux

Для сборки конечного приложения в заголовочном файле \include__can_defines.h следует выбрать тип операционной системы Linux: #define CAN_OS_LINUX и установить

параметр режима сборки конечного приложения CAN_APPLICATION_MODE. При необходимости можно переопределить другие конфигурационные параметры, см. «Параметры режимов и сборки CANopen приложения».

Компиляция приложения запускается командой make с одним из параметров: ”make canmaster” для master приложения, ”make canslave” для slave приложения или ”make cantest” для компиляции тестового приложения. При этом в файле Make.vars следует при необходимости скорректировать путь к заголовочным и библиотечным файлам драйвера CHAI. В результате компиляции формируется исполняемый модуль приложения *canapp.

Технология реализации функций и протоколов библиотеки

Фильтр входящих кадров CAN контроллера

Фильтрация входящих CAN кадров производится по битовой маске идентификатора CAN–ID. Фильтр пропускает лишь те кадры, в которых некоторые биты имеют определенное фиксированное значение. Поскольку все CAN узлы должны принимать NMT кадры с идентификатором равным нулю, при фильтрации необходимо пропускать все CAN–ID, в которых значение любого бита равно нулю. Таким образом, при одно-уровневой фильтрации не удается избавиться от кадров, где биты идентификатора могут принимать значения как 0, так и 1, а значит эффективность такой фильтрации зависит от номера CAN узла. Так, узел с номером 127 будет принимать все CAN кадры, поскольку должен обрабатывать как NMT запросы с идентификатором равным нулю, так и адресованные самому узлу кадры со значением семи младших бит CAN–ID (код номера узла для предопределенного распределения идентификаторов) равными 1.

Двухуровневая фильтрация лишена этого недостатка. Здесь имеется возможность отдельно отфильтровать широковещательные кадры с нулевым значением поля номера узла идентификатора (NMT, SYNC, TIME) и со значением семи младших бит CAN–ID, соответствующих номеру CAN узла. Таким образом, для предопределенного распределения идентификаторов отбираются лишь кадры, предназначенные данному узлу.

Для протокола EN50325-5 может быть использована трехуровневая фильтрация, при условии ее поддержки драйвером канального уровня. Третий масочный фильтр настраивается на прием кадров со значениями идентификаторов, отличных от предопределенных.

Методы обработки CAN–ID входящих кадров

Поддерживаются два способа обработки CAN идентификаторов входящих кадров: динамический и статический. Динамический метод требует заметно меньше памяти, но не столь эффективен по быстродействию, как статический. Для динамического метода создается не упорядоченный массив записей, содержащих значения CAN–ID и соответствующих им индексов коммуникационного раздела объектного словаря. При получении CAN кадра производится линейный поиск в этом массиве индекса, соответствующего поступившему идентификатору. Полное число обрабатываемых динамическим методом CAN идентификаторов определяется параметрами CAN_NOF_RECVCANID_MASTER (для master) и CAN_NOF_RECVCANID_SLAVE (для slave) в модуле \include__can_defines.h. Динамический метод сохраняет эффективность, когда полное число обрабатываемых CAN–ID не превышает 50..100, в зависимости от производительности процессора. Статический метод используется только для CANopen мастера и 11-битовых CAN–ID. Он создает массив, размер которого соответствует максимально возможному числу идентификаторов, сопоставляемых индексам коммуникационного раздела объектного словаря. Метод обработки CAN–ID для мастера определяется параметром CAN_MASTER_RECVCANID_METHOD в модуле \include__can_defines.h. Фильтр входящих кадров CAN контроллера устанавливается только для динамического метода. Для CANopen узла (slave) также используется только динамический метод, ввиду малого числа конфигурируемых CAN идентификаторов.

Идентификаторы ограниченного использования

Из всего набора идентификаторов ограниченного использования не зависимо от настроек обрабатываются идентификаторы NMT объектов: NMT (значение CAN–ID равно 0_h)

и NMT Error Control (значения CAN–ID в диапазоне $701_h..77F_h$). Кроме того, при активации LSS протоколов идентификаторы $7E5_h$ (запрос от LSS master) и $7E4_h$ (ответ от LSS slave) также обрабатываются непосредственно. Назначение этих идентификаторов другим объектам не позволит последним их использовать, поскольку соответствующие CAN–ID перехватываются до обработки любых конфигурируемых идентификаторов.

Начиная с версии 2.2 введена проверка и запрет идентификаторов ограниченного использования в любых конфигурируемых COB-ID. Для SDO объектов значения идентификаторов могут находиться только в допустимых диапазонах. Вместе с тем, контроль за использованием других конфигурируемых идентификаторов не осуществляется.

Реализация объектного словаря

Записи объектных словарей реализованы статически. Это дает возможность асинхронного доступа к словарю, например, при обмене PDO. Пример реализации объектного словаря slave для профиля тестового устройства приведен в модуле \slave__obdms_slave_test.h.

В задачу библиотеки не входит поддержка в CANopen мастере полной инфраструктуры работы со словарями всех slave устройств. В качестве одной из возможных моделей реализации доступа к словарям slave устройств библиотека предлагает отображение в мастере (клиенте) разделов словаря, определяющих объекты прикладных профилей. Это дает возможность непосредственного обмена содержимым записей как с помощью SDO, так и PDO протоколов. Коммуникационные разделы объектного словаря клиента и сервера не требуют отображения и реализованы независимо. Для целей тестирования к библиотеке подключаются соответствующие модули статического отображение словарей. Пример реализации мастер–отображения объектного словаря для профиля тестового устройства приведен в модуле \master__obdms_master_test.h. В качестве примеров реализации и работы с объектными словарями могут также использоваться адаптированные версии библиотеки и версия DLL мастера.

Реализация SDO протоколов

В ускоренном SDO протоколе передается до четырех байт данных, заключенных в единственный CAN кадр: дополнительная буферизация в этом случае не требуется. Сегментированный протокол осуществляет обмен данными с буферизацией как на передающей, так и на принимающей стороне. При инициализации протокола передающая сторона считывает соответствующую запись из объектного словаря в динамический буфер, а принимающая выделяет буфер необходимого размера. Данные обновляются в объектном словаре принимающей стороны только после успешного завершения всего цикла передачи. Максимальный размер записи объектного словаря в байтах, передаваемой посредством сегментированного SDO протокола, определяется параметром CAN_SIZE_MAXSDOMEM. Этот параметр используется в специальной сигнало-безопасной функции динамического выделения памяти для определения максимального размера буфера. Блочный протокол использует буферизацию только на стороне сервера и лишь в случае, когда все данные можно разместить в динамическом буфере. Но, как правило, передача данных производится непосредственно между записями объектного словаря передающей и принимающей сторон. Для этого обеспечивается доступ к соответствующим записям словаря посредством байтового указателя. Блочный протокол гарантирует состоятельность данных объектного словаря принимающей стороны только после успешного завершения всего цикла обмена. В противном случае данные соответствующей записи словаря использоваться не должны.

Реализация LSS протоколов

LSS протокол активируется когда slave устройство обнаруживает, что ему присвоен номер CAN узла, равный 255 (не сконфигурированное CANopen устройство). При этом оно переходит в состояние ожидания (LSS waiting). LSS slave устройство становится сконфигурированным после записи нового значения номера CAN узла в диапазоне от 1 до 127 в энергонезависимую память. Для осуществления такой записи используется LSS протокол "Store configuration". LSS Fastscan протокол активен только для LSS slave устройств с номером CAN узла 255 (не сконфигурированное устройства).

Идентификаторы LSS протокола обрабатываются не зависимо от настройки идентификаторов любых других коммуникационных CANopen объектов. Кроме того, сконфигурированное LSS устройство отрабатывает две NMT команды: Reset Node и Reset Communication. После их выполнения такое устройство переходит в штатное пред-операционное NMT состояние. Никакие другие коммуникационные CANopen объекты в LSS устройстве не используются. Однако, для локализации возможных ошибок приложения инициализация коммуникационных объектов CANopen производится со значением номера CAN узла, равным нулю.

Модуль сохранения параметров в энергонезависимой памяти

В библиотечном модуле can_obj_re_store.c фактическое хранение параметров осуществляется в статических массивах данных, размещаемых в оперативной памяти. Состоятельность данных контролируется 16–разрядным CRC кодом. При переносе программы модуля на микроконтроллерную платформу эти массивы должны быть заменены на соответствующие адреса энергонезависимой памяти. Кроме того, для работы с такой памятью следует использовать API применяемой платформы.

Начиная с версии библиотеки 2.0 объекты сохранения/восстановления параметров (1010_h, 1011_h) поддерживают 6 субиндексов. Дополнительные субиндексы имеют следующее назначение:

1010_h $sub4_h$: Сохранения каких–либо параметров не осуществляется.

1010_h $sub5_h$: Сохранение действующего номера CAN узла устройства.

1010_h $sub6_h$: Сохранение установленного индекса битовой скорости устройства.

$1011_h sub4_h$: Восстановление значения по умолчанию для параметров: 1005_h, 1012_h, 1014_h, $1400_h sub1_h$, $1401_h sub1_h$, $1402_h sub1_h$, $1403_h sub1_h$, $1800_h sub1_h$, $1801_h sub1_h$, $1802_h sub1_h$, $1803_h sub1_h$. Значения по умолчанию для этих параметров задают предопределенное распределение идентификаторов соответствующих коммуникационных объектов. При этом учитывается номера CAN узла устройства.

1011_h $sub5_h$: Восстановление номера CAN узла устройства (значения по умолчанию).

1011_h $sub6_h$: Восстановление индекса битовой скорости устройства (значения по умолчанию).

Реализация безопасного протокола EN50325-5

Протокол EN50325-5 обеспечивает полную совместимость со стандартом CiA 301. В его реализации используется отдельный сегмент CAN идентификаторов и собственный набор индексов объектного словаря для коммуникаций. Поэтому устройство с поддержкой EN50325-5 может одновременно работать по обоим коммуникационным протоколам. При работе с безопасным протоколом следует учитывать, что он формирует дополнительный сетевой трафик с высоким приоритетом CAN кадров и жесткими временны*ы*ми требованиями. Соответственно, число узлов сети, которые осуществляют передачу данных по безопасному протоколу должно быть ограничено.

Программный код EN50325-5 включен в состав адаптированных версий библиотеки. Описание реализации протокола приведено в документе CANopen_slave.pdf.

Типы и структуры данных, используемые в API библиотеки

Типы данных библиотеки

Обозначение	Тип данных	Описание
canbyte	**unsigned8**	Без-знаковое целое 8 бит.
cannode	**unsigned8**	Без-знаковое целое 8 бит, номер CAN узла.
canindex	**unsigned16**	Без-знаковое целое 16 бит, индекс объектного словаря.
cansubind	**unsigned8**	Без-знаковое целое 8 бит, субиндекс объектного словаря.
canlink	**unsigned16**	Без-знаковое целое 16 бит, CAN идентификатор канального уровня для 11 битового CAN–ID.
canlink	**unsigned32**	Без-знаковое целое 32 бита, CAN идентификатор канального уровня для 29 битового CAN–ID. Не используется в CANopen.

Типы данных драйвера канального уровня CHAI

Обозначение	Описание
_u8	Без-знаковое целое 8 бит.
_s8	Целое 8 бит со знаком.
_u16	Без-знаковое целое 16 бит.
_s16	Целое 16 бит со знаком.
_u32	Без-знаковое целое 32 бита.
_s32	Целое 32 бита со знаком.

Структуры данных API

struct sdoixs {
canindex sdoind индекс коммуникационного SDO параметра клиента либо сервера (объекты 1200_h .. $12FF_h$).
canindex index индекс прикладного объекта.
cansubind subind субиндекс прикладного объекта.
};
Структура **sdoixs** определяет коммуникационный объект SDO протокола, а также индекс и субиндекс прикладного объекта (мультиплексор SDO протокола).

struct sdostatus {
int16 state статус во время и после завершения SDO транзакции клиента.
unsigned32 abortcode SDO аборт код, если по завершении транзакции **state** принимает значение CAN_TRANSTATE_SDO_SRVABORT.
};
Структура **sdostatus** размещает информацию о статусе SDO транзакции клиента

struct sdocltappl {
unsigned8 operation базовый режим передачи SDO (upload / download).
cannode node номер узла SDO сервера.

unsigned32 datasize размер данных в байтах.
canbyte *datapnt байтовый указатель на локальный буфер.
struct sdoixs si структура SDO индексов.
struct sdostatus ss статус SDO транзакции.
};
Структура **sdocltappl** служит для взаимодействия с приложением клиента и используется при обмене данными с помощью SDO протокола.

struct canframe {
unsigned32 id CAN–ID.
unsigned8 data[8] поле данные CAN кадра.
unsigned8 len реальная длина данных (от 0 до 8).
unsigned16 flg битовые флаги CAN кадра. Бит 0 – RTR, бит 2 – EFF.
unsigned32 ts временн*а*я метка получения CAN кадра в микросекундах.
};
Структура **canframe** размещает CAN кадр канального уровня. Ее определение содержится в заголовочном файле CAN драйвера CHAI (структура **canmsg_t**).

Глобальные данные

Обозначение	Тип данных	Описание
can_netchan	unsigned8	Номер активной CAN сети (от 0 до 7).
bitrate_index	unsigned8	Индекс битовой скорости CAN сети.
node_id	unsigned8	Номер CAN узла (от 1 до 127).
node_state	unsigned8	NMT состояние CAN узла.
lss_state	unsigned8	Состояние LSS протокола (off, waiting, configuration).

Размещение модулей библиотеки

Размещение модулей библиотеки приведено относительно «корневой» директории CANopen.

master директория. Модули CANopen для режима master.

- can_client.c – поддержка полных SDO транзакций клиента.
  can_clt_block.h – поддержка SDO транзакций клиента для блочного протокола.
- can_cltrans.c – поддержка базовых SDO транзакций клиента (запрос клиента и ответ сервера).
- can_obdclt.c – диспетчер доступа к компонентам объектных словарей клиента (master).
- can_obdsdo_client.c – объектный словарь SDO параметров клиента.
- can_test_driver.c – замыкающий (loopback) CAN драйвер для тестового режима.

Следующие модули этой директории могут редактироваться пользователем:

- __can_test_application.c – операции клиента и отображение словаря тестового устройства.
- __obd_mans_master.c – диспетчер доступа к отображениям объектных словарей slave устройств в мастере.
  __obdms_master_*.h – отображения объектных словарей прикладных профилей slave устройств в мастере.

slave директория. Модули CANopen для режима slave.

- can_lss_slave.c – поддержка LSS slave протоколов (службы установки уровня).
- can_obdsdo_server.c – диспетчер модулей объектного словаря SDO параметров сервера.
  can_obdsdo_server_default.h – объектный словарь единственного SDO параметра сервера, используемого по умолчанию.
  can_obdsdo_server_num.h – объектный словарь нескольких SDO параметров сервера.
- can_obj_device.c – объектный словарь описания устройства.
- can_obdsrv.c – диспетчер доступа к компонентам объектных словарей сервера (slave).
- can_server.c – диспетчер модулей полных SDO транзакций сервера.
  can_server_block.h – поддержка SDO транзакций сервера для блочного протокола.
  can_server_common.h – общие функции модулей полных SDO транзакций.
  can_server_min.h – поддержка полных SDO транзакций сервера для единственного SDO параметра по умолчанию.
  can_server_standard.h – поддержка полных SDO транзакций сервера для нескольких SDO параметров.

Следующие модули этой директории могут редактироваться пользователем:

- __can_devices.c – диспетчер описания прикладных профилей.
  __can_device_*.h – описания различных прикладных профилей.
- __obd_mans_slave.c – диспетчер объектных словарей прикладных профилей.
  __obdms_slave_*.h – объектные словари прикладных профилей.

common директория. Общие модули CANopen.

Директория \pdomapping содержит объектный словарь PDO отображения.

- can_backinit.c – функции (пере)инициализации CAN устройства, диспетчер таймера и CANopen монитор.
- can_canid.c – диспетчер модулей динамической или статической обработки CAN–ID.
  can_canid_dynamic.h – поддержка динамических CAN–ID и масочного фильтра входящих CAN кадров.
  can_canid_static.h – поддержка статических CAN–ID.
- can_globals.c – определения внешних (глобальных) переменных и структур данных.
- can_inout.c – взаимодействие с драйвером CAN сети, ввод/вывод CAN кадров канального уровня, первичный разбор идентификаторов принимаемых кадров.
- can_led_indicator.c – светодиодная индикация состояния устройства.

- can_lib.c – функции общего назначения: подсчет CRC, преобразование данных и т.п.
- can_malloc.c – специализированная сигнало-безопасная функция выделения динамической памяти.
- can_nmt_master.c – NMT master.
- can_nmt_slave.c – NMT slave.
- can_obj_deftype.c – объекты определения типов данных.
- can_obj_emcy.c – объекты срочных сообщений EMCY.
- can_obj_errors.c – объекты ошибок.
- can_obj_err_behaviour.c – объект, определяющий поведение CANopen устройства при возникновении серьезных ошибок.
- can_obj_re_store.c – объекты сохранения/восстановления параметров в энергонезависимой памяти.
- can_obj_sync.c – объекты синхронизации SYNC.
- can_obj_time.c – объект временной метки.
- can_pdo_map.c – диспетчер модулей динамического или статического PDO отображения.
  can_pdo_map_dynamic_bit.h – модуль динамического бит–отображения PDO.
  can_pdo_map_dynamic_byte.h – модуль динамического байт–отображения PDO.
  can_pdo_map_static.h – модуль статического байт–отображения PDO.
- can_pdo_obd.c – поддержка объектного словаря коммуникационных PDO параметров.
- can_pdo_proc.c – обработка принимаемых и передаваемых PDO кадров.
- can_sdo_proc.c – обработка принимаемых и передаваемых SDO кадров.

Следующие модули этой директории могут редактироваться пользователем:

- __can_events.c – обработчики CANopen событий (EMCY, errors, и др).
- __can_init.c – определение номера CAN узла и скорости CAN сети.
- pdomapping__map__static.h – конфигурирование объектов статического PDO отображения.
- pdomapping__map_recv_*_*.h , pdomapping__map_tran_*_*.h – конфигурирование принимаемых и передаваемых объектов динамического PDO отображения.

include директория. Модули определений и прототипов.

- can_defines.h – определение параметров (констант), специфических для CANopen.
- can_defunc.h – базовый модуль прототипов функций. Включает общие внутренние для библиотеки функции.
- can_defunc_master.h – прототипы внутренних функций для master.
- can_defunc_nmt.h – прототипы NMT функций.
- can_defunc_slave.h – прототипы внутренних функций для slave.
- can_genhead.h – основной модуль заголовков и подключений.
- can_globals.h – внешние (глобальные) переменные библиотеки.
- can_header.h – базовый заголовочный модуль.
- can_macros.h – определение макросов CANopen библиотеки.
- can_structures.h – определение структур данных.
- can_typedefs.h – определение типов данных.
- can_user_api_call.h – прототипы API функций, вызываемых приложением пользователя.
- can_user_api_edit.h – прототипы API функций, вызываемых CANopen событиями.

Следующие модули этой директории могут редактироваться пользователем:

- __can_defines.h – определение конфигурационных параметров (констант).
- __can_defunc_master.h – функции отображения в мастере объектных словарей slave.
- __can_node_id.h – задаются номер CAN узла и серийный номер устройства.

Корневая директория CANopen.

Функции, зависящие от операционной системы: CANopen таймер, временные задержки, критические секции и др.

- can_system_linux.h – модуль системно–зависимых функций для ОС Linux.
- can_system_windows.h – модуль системно–зависимых функций для ОС Windows.

Следующие модули этой директории могут редактироваться пользователем:

- __can_main.c – содержит запускаемую на выполнение функцию main(...) и главный цикл библиотеки.
- __can_system.c – диспетчер подключения системно–зависимых модулей.

Функциональное назначение модулей библиотеки

Модули библиотеки могут принадлежать нескольким функциональным группам.

Модули работы с CAN сетью на канальном уровне

- can_canid.c – диспетчер модулей динамической или статической обработки CAN–ID.
  can_canid_dynamic.h – поддержка динамических CAN–ID и масочного фильтра входящих кадров CAN контроллера.
  can_canid_static.h – поддержка статических CAN–ID.
- can_inout.c – взаимодействие с драйвером CAN сети, ввод/вывод CAN кадров канального уровня, первичный разбор идентификаторов принимаемых кадров.
- can_test_driver.c – замыкающий (loopback) CAN драйвер для тестового режима.

Модули поддержки SDO транзакций

Обмен данными в SDO протоколе инициируется и осуществляется под управлением клиента. Поэтому SDO транзакции клиента реализованы по двухуровневой схеме. Базовая транзакция осуществляет передачу серверу одного CAN кадра, ожидание и прием ответа. Полная SDO транзакция клиента контролирует весь цикл обмена данными. Сервер SDO протокола отвечает на запросы клиента, поэтому в нем не предусмотрено явного выделения базовой и полной транзакций. В то же время, сервер отслеживает весь ход обмена данными в SDO протоколе.

- can_client.c – поддержка полных SDO транзакций клиента.
  can_clt_block.h – поддержка SDO транзакций клиента для блочного протокола.
- can_cltrans.c – поддержка базовых SDO транзакций клиента.
- can_server.c – диспетчер модулей полных SDO транзакций сервера.
  can_server_block.h – поддержка SDO транзакций сервера для блочного протокола.
  can_server_common.h – общие функции модулей полных SDO транзакций.
  can_server_min.h – поддержка полных SDO транзакций сервера для единственного SDO параметра по умолчанию.
  can_server_standard.h – поддержка полных SDO транзакций сервера для нескольких SDO параметров.

Модули сборки, разборки и обработки CANopen объектов

- can_pdo_proc.c – обработка принимаемых и передаваемых PDO кадров.
- can_sdo_proc.c – обработка принимаемых и передаваемых SDO кадров.
- can_obj_emcy.c – формирование объекта EMCY.
- can_nmt_master.c – формирование, прием и передача NMT master объектов.
- can_nmt_slave.c – формирование, прием и передача NMT slave объектов.
- can_pdo_map.c – диспетчер модулей динамического или статического PDO отображения.
  can_pdo_map_dynamic_bit.h – сборка, разборка, активация динамического бит–отображения PDO.
  can_pdo_map_dynamic_byte.h – сборка, разборка, активация динамического байт–отображения PDO.
  can_pdo_map_static.h – сборка, разборка, активация статического байт–отображения PDO.

Объектный словарь коммуникационного профиля

- can_obdclt.c – диспетчер доступа к компонентам объектных словарей клиента (master).
- can_obdsrv.c – диспетчер доступа к компонентам объектных словарей сервера (slave).
- can_obj_device.c – поддержка объектного словаря описания устройства.
- __can_devices.c – диспетчер описания прикладных профилей.
  __can_device_*.h – описания различных прикладных профилей (объекты 1000_h, 1002_h, 1008_h, 1009_h, $100A_h$, 1018_h).
- can_obdsdo_client.c – объектный словарь SDO параметров клиента (объекты $1280_h..12FF_h$).
- can_obdsdo_server.c – диспетчер модулей объектного словаря SDO параметров сервера.
  can_obdsdo_server_default.h – объектный словарь SDO параметра сервера по умолчанию (объект 1200_h).
  can_obdsdo_server_num.h – объектный словарь нескольких SDO параметров сервера (объекты $1200_h..127F_h$).
- can_nmt_master.c – NMT master (объекты 1016_h, $100C_h$, $100D_h$).
- can_nmt_slave.c – NMT slave (объекты 1017_h, $100C_h$, $100D_h$).
- can_obj_deftype.c – объект определения типов данных ($0001_h..0007_h$).
- can_obj_emcy.c – объект EMCY (1014_h, 1015_h).
- can_obj_errors.c – объект ошибок (1001_h, 1003_h).
- can_obj_err_behaviour.c – объект, определяющий поведение CAN устройства при возникновении серьезных ошибок (1029_h). Только для NMT slave.
- can_obj_re_store.c – объект сохранения/восстановления параметров в энергонезависимой памяти (1010_h, 1011_h).
- can_obj_sync.c – объект синхронизации SYNC (1005_h, 1006_h, 1007_h, 1019_h).
- can_obj_time.c – объект временной метки (1012_h).
- can_pdo_obd.c – поддержка объектного словаря коммуникационных PDO параметров (объекты $1400_h..15FF_h$, $1800_h..19FF_h$).
- can_pdo_map.c – диспетчер модулей динамического или статического PDO отображения (объекты $1600_h..17FF_h$, $1A00_h..1BFF_h$).
  can_pdo_map_dynamic_bit.h – модуль динамического бит–отображения PDO.
  can_pdo_map_dynamic_byte.h – модуль динамического байт–отображения PDO.
  can_pdo_map_static.h – модуль статического байт–отображения PDO.
  \pdomapping__map__static.h – конфигурирование объектов статического PDO отображения.
  \pdomapping__map_recv_*_*.h , pdomapping__map_tran_*_*.h – конфигурирование принимаемых и передаваемых объектов динамического PDO отображения.

Объектный словарь прикладных профилей

- __can_test_application.c – операции клиента и отображение словаря тестового устройства.
- __obd_mans_master.c – диспетчер доступа к отображениям объектных словарей slave устройств в мастере.
  __obdms_master_*.h – отображения объектных словарей прикладных профилей slave устройств в мастере.
- __obd_mans_slave.c – диспетчер объектных словарей прикладных профилей slave.
  __obdms_slave_*.h – объектные словари прикладных профилей slave.

Модули общего назначения

- can_globals.c – определения внешних (глобальных) переменных и структур данных.
- can_lib.c – функции общего назначения: подсчет CRC, преобразование данных и т.п.
- can_malloc.c – специализированная сигнало-безопасная функция выделения динамической памяти.

Модули инициализации и обработки событий

- can_backinit.c – функции (пере)инициализации CAN устройства, диспетчер таймера и CANopen монитор.
- __can_events.c – обработчики CANopen событий (EMCY, errors, и др).
- __can_init.c – определение номера CAN узла и скорости CAN сети.

Прочие модули

- can_led_indicator.c – светодиодная индикация состояния устройства.
- can_lss_slave.c – поддержка LSS slave протоколов (службы установки уровня).
- __can_main.c – содержит запускаемую на выполнение функцию main(...) и главный цикл библиотеки.
- __can_system.c – диспетчер подключения системно–зависимых модулей.
  can_system_linux.h – модуль системно–зависимых функций для ОС Linux.
  can_system_windows.h – модуль системно–зависимых функций для ОС Windows.

Взаимодействие библиотеки с API драйвера канального уровня

CANopen библиотека подключается к канальному уровню CAN с использованием API драйвера CHAI. При этом задействованы только базовые функции драйвера, присутствующие во всех его версиях. В данном разделе поясняется назначение функций драйвера, что позволяет облегчить его разработку для других целевых платформ.

_s16 CiInit(void);
Осуществляет начальную инициализацию CAN контроллера на аппаратном уровне.
Инициализирует структуры данных драйвера и всех CAN каналов.
Функция выполняется однократно при запуске CAN устройства.
Вызывается в модуле can_backinit.c.
Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

_s16 CiOpen(_u8 chan, _u8 flags);
Инициализирует канал контроллера **chan** в не блокирующем режиме с возможностью обработки 11-битовых CAN идентификаторов. Устанавливает аппаратные режимы канала контроллера. Инициализирует структуры данных этого канала.
Вызывается в модуле can_backinit.c.
Параметры:
- **chan** – номер канала CAN контроллера (считаются с 0).
- **flags** – задает типы обрабатываемых CAN идентификаторов (11-битовые и/или 29-битовые).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

_s16 CiClose(_u8 chan);
Закрывает канал **chan**. Запрещает прерывания, сбрасывает регистры, удаляет обработчики сигналов. Сбрасывает фильтр входящих CAN кадров. Последовательность вызова функций CiClose(…) → CiOpen(…) выполняет пере-инициализацию канала CAN контроллера.
Вызывается в модуле can_backinit.c.
Параметры:
- **chan** – номер канала CAN контроллера (считаются с 0).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

_s16 CiStart(_u8 chan);
Переводит канал контроллера **chan** в активное состояние, разрешая аппаратные прерывания.
Вызывается в модулях can_backinit.c, can_canid_dynamic.h.
Параметры:
- **chan** – номер канала CAN контроллера (считаются с 0).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

_s16 CiStop(_u8 chan);
Переводит канал **chan** в не активное состояние, запрещая аппаратные прерывания.
Вызывается в модулях can_backinit.c, can_canid_dynamic.h.
Параметры:
- **chan** – номер канала CAN контроллера (считаются с 0).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

_s16 CiSetFilter(_u8 chan, _u32 acode, _u32 amask);
Устанавливает одно-уровневый масочный фильтр входящих кадров CAN контроллера.
_s16 CiSetDualFilter(_u8 chan, _u32 acode0, _u32 amask0, _u32 acode1, _u32 amask1);
Устанавливает двух-уровневый масочный фильтр входящих кадров CAN контроллера.
_s16 CiSetFilter_1(_u8 chan, _u32 acode, _u32 amask);
_s16 CiSetFilter_2(_u8 chan, _u32 acode, _u32 amask);
_s16 CiSetFilter_3(_u8 chan, _u32 acode, _u32 amask);
Функции используются для установки трех-уровневого масочного фильтра входящих кадров CAN контроллера
Масочный фильтр может быть реализован на аппаратном уровне, если такая возможность поддерживается CAN контроллером. Производительность современных микроконтроллеров достаточна для реализации такого фильтра в драйвере и на программном уровне.
Вызываются в модуле can_canid_dynamic.h.
Параметры:
- **chan** – номер канала CAN контроллера (считаются с 0).
- **acode, acode0, acode1** – требуемые значения бит для фильтров.
- **amask, amask0, amask1** – битовая маска фильтров (1 — значение соответствующего бита **acode** учитывается, 0 — игнорируется, то есть бит может принимать любое значение).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

_s16 CiSetBaud(_u8 chan, _u8 bt0, _u8 bt1);
Устанавливает битовую скорость CAN сети для канала контроллера **chan**,
Вызывается в модуле can_backinit.c.
Параметры:
- **chan** – номер канала CAN контроллера (считаются с 0).
- **bt0, bt1** – коды скорости, значения которых зависит от типа CAN контроллера.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

_s16 CiWrite(_u8 chan, canmsg_t *mbuf, _s16 cnt);
_s16 CiWrite(_u8 chan, canmsg_t *mbuf);
Записывает в буфер контроллера **chan** (или в очередь драйвера) один кадр данных канального уровня. Запись производится в не блокирующем режиме. Для улучшения динамических характеристик библиотеки рекомендуется устанавливать нулевое значение таймаута при записи кадров. Для прикладных функций CANopen библиотека обеспечивает сигнало–безопасную, повторно–входимую запись CAN кадров в программный кэш.
Вызывается в модуле can_inout.c.
Параметры:
- **chan** – номер канала CAN контроллера (считаются с 0).
- ***mbuf** – указатель на структуру CAN кадра канального уровня.
- **cnt** – число кадров для записи (при наличие в API). Для CANopen библиотеки всегда равно 1.

Возвращаемые значения: нормальное завершение = 1 (число фактически записанных CAN кадров); ошибка <= 0.

_s16 CiRead(_u8 chan, canmsg_t *mbuf, _s16 cnt);
_s16 CiRead(_u8 chan, canmsg_t *mbuf);
Считывает из буфера контроллера **chan** (или из очереди драйвера) один кадр данных канального уровня для обработки CANopen библиотекой. Вызывается из обработчика события приема CAN кадра.
Вызывается в модуле can_inout.c.
Параметры:

- **chan** – номер канала CAN контроллера (считаются с 0).
- ***mbuf** – указатель на структуру CAN кадра канального уровня.
- **cnt** – число кадров для чтения (при наличие в API). Для CANopen библиотеки всегда равно 1.

Возвращаемые значения: нормальное завершение = 1 (число фактически прочитанных CAN кадров); ошибка <= 0.

_s16 CiSetCB(_u8 chan, _u8 ev, void (*ci_handler) (_s16));
Регистрирует обработчик сигналов (событий) приема CAN кадров для канала контроллера **chan**. Обработчик является сигнало-безопасным. Его вызов возможен непосредственно из аппаратных прерываний CAN контроллера. Обработчик обеспечивает последовательное чтение кадров, поступающих в буфер контроллера (или в очередь драйвера) в том числе в процессе обработки текущего CAN кадра (повторно-входимый режим). Следует учитывать, что при каждом вызове обработчика выполняется значительный объем программного кода.
Вызывается в модуле can_backinit.c.
Параметры:

- **chan** – номер канала CAN контроллера (считаются с 0).
- **ev** – событие, для которого устанавливается обработчик (прием CAN кадра).
- ***ci_handler** – указатель на функцию обработки принятых кадров can_read_handler(…), которая размещается в модуле can_inout.c.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

_s16 CiSetCB(_u8 chan, _u8 ev, void (*ci_handler) (_s16));
Регистрирует обработчик сигналов (событий) возникновения ошибок для канала контроллера **chan**. Обработчик является сигнало-безопасным. Его вызов возможен непосредственно из аппаратных прерываний CAN контроллера. При наложении обращений к обработчику (повторно-входимый режим) возможна потеря записей в списке предопределенных ошибок (объект 1003_h), но в любом случае информация сохраняется в регистре ошибок (объект 1001_h). Следует учитывать, что при каждом вызове обработчика выполняется значительный объем программного кода.
Вызывается в модуле can_backinit.c.
Параметры:

- **chan** – номер канала CAN контроллера (считаются с 0).
- **ev** – событие, для которого устанавливается обработчик (сигнал возникновения ошибки).
- ***ci_handler** – указатель на функцию обработки ошибок consume_controller_error(…), которая размещается в модуле __can_event.c.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

void ci_propagate_sigs(void);
Пропагатор (функция распространения) сигналов драйвера. В случае, если драйвер не обеспечивает асинхронную доставку каких-либо входящих сигналов (событий) библиотеке, пропагатор должен быть включен в главный цикл программы. При этом время задержки до начала обработки события, например, принятого CAN кадра, зависит от полной длительности главного цикла.

Раздел объектного словаря для коммуникаций

Представлены все коммуникационные объекты, поддерживаемые библиотекой CANopen. Размещение модулей приводится относительно «корневой» директории библиотеки.

NMT объекты

- **$100C_h$** Guard time
  Охранное время в миллисекундах. Модуль \common\can_nmt_master.c для NMT master и \common\can_nmt_slave.c для NMT slave. Master объект представлен массивом.
- **$100D_h$** Life time factor
  Множитель времени жизни. Модуль \common\can_nmt_master.c для NMT master и \common\can_nmt_slave.c для NMT slave. Master объект представлен массивом.
- **1016_h** Consumer heartbeat time
  Период сердцебиения потребителя в миллисекундах. Модуль \common\can_nmt_master.c. Число субиндексов этого объекта определяется параметром CAN_NOF_NODES. Протокол сердцебиения имеет более высокий приоритет, чем протокол охраны узла. Период сердцебиения потребителя инициализируется для каждого узла значением по умолчанию CAN_HBT_CONSUMER_MS. Для того, чтобы активировать протокол охраны узла, периоды сердцебиения как поставщика, так и потребителя должны быть обнулены.
- **1017_h** Producer heartbeat time
  Период сердцебиения поставщика в миллисекундах. Модуль \common\can_nmt_slave.c. Протокол сердцебиения имеет более высокий приоритет, чем протокол охраны узла. Период сердцебиения поставщика инициализируется значением по умолчанию CAN_HBT_PRODUCER_MS. Для того, чтобы активировать протокол охраны узла, периоды сердцебиения как поставщика, так и потребителя должны быть обнулены.

Объекты, представленные в master и slave

- **1005_h** COB–ID SYNC message
  COB–ID объекта синхронизации SYNC. Модуль \common\can_obj_sync.c
- **1006_h** Communication cycle period
  Период объекта синхронизации в микросекундах. Модуль \common\can_obj_sync.c
- **1007_h** Synchronous window length
  Длительность окна синхронизации в микросекундах (временно*е* окно для обработки синхронных PDO). Модуль \common\can_obj_sync.c.
- **1012_h** COB–ID time stamp object
  COB–ID объекта временно*й* метки TIME. Модуль \common\can_obj_time.c
- **1019_h** Synchronous counter overflow value
  Значение переполнения для SYNC счетчика. Модуль \common\can_obj_sync.c.
- **1029_h** Error behaviour object
  Объект, определяющий поведение CANopen устройства при возникновении серьезных ошибок. Поддерживается только для NMT slave. Модуль \common\can_obj_err_behaviour.c.
- **1400_h..$15FF_h$** RPDO communication parameter
  Коммуникационные параметры принимаемых PDO (RPDO). Объектный словарь, формирующий коммуникационные параметры до 512 RPDO, содержится в модуле \common\can_pdo_obd.c. Фактическое число принимаемых PDO определяется параметрами CAN_NOF_PDO_RECV_SLAVE для slave и CAN_NOF_PDO_RECV_MASTER для master. Обработка операций RPDO протокола производится в модуле \common\can_pdo_proc.c

- **1600$_h$..17FF$_h$** RPDO mapping parameter
  Параметры отображения (mapping) принимаемых PDO (RPDO). Для динамического бит-ориентированного PDO отображения объектный словарь, определяющий параметры отображения до 512 RPDO, содержится в модуле \common\can_pdo_map_dynamic_bit.h. По мере необходимости этот модуль подгружает mapping–определения из поддиректории \pdomapping. Диспетчер определений размещается в модуле \pdomapping\can_mappdo_main.h, а сами mapping–определения агрегатированы по 32 RPDO в каждом \pdomapping__map_recv_*_*.h файле.
  Словарь динамического байт-ориентированного PDO отображения формируется в модуле \common\can_pdo_map_dynamic_byte.h.
  Параметры статического PDO отображения определяются в модулях \common\can_pdo_map_static.h и \pdomapping__map__static.h.
- **1800$_h$..19FF$_h$** TPDO communication parameter
  Коммуникационные параметры передаваемых PDO (TPDO). Объектный словарь, определяющий коммуникационные параметры до 512 TPDO, содержится в модуле \common\can_pdo_obd.c. Фактическое число передаваемых PDO определяется параметрами CAN_NOF_PDO_TRAN_SLAVE для slave и CAN_NOF_PDO_TRAN_MASTER для master. Обработка операций TPDO протокола производится в модуле \common\can_pdo_proc.c
- **1A00$_h$..1BFF$_h$** TPDO mapping parameter
  Параметры отображения (mapping) передаваемых PDO (TPDO). Для динамического бит-ориентированного PDO отображения объектный словарь, определяющий параметры отображения до 512 TPDO, содержится в модуле \common\can_pdo_map_dynamic_bit.h. По мере необходимости этот модуль подгружает mapping–определения из поддиректории \pdomapping. Диспетчер определений размещается в модуле \pdomapping\can_mappdo_main.h, а сами mapping–определения агрегатированы по 32 TPDO в каждом \pdomapping__map_tran_*_*.h модуле.
  Словарь динамического байт-ориентированного PDO отображения формируется в модуле \common\can_pdo_map_dynamic_byte.h.
  Параметры статического PDO отображения определяются в модулях \common\can_pdo_map_static.h и \pdomapping__map__static.h.

Master объекты

- **1280$_h$..12FF$_h$** SDO client parameter
  SDO параметры клиента. Объектный словарь, определяющий до 128 SDO параметров клиента, содержится в модуле \master\can_obdsdo_client.c. Фактическое число клиентских SDO определяется параметром CAN_NOF_NODES (число узлов CAN сети) в модуле \include__can_defines.h. Обработка операций SDO протокола производится в модуле \common\can_sdo_proc.c

Slave объекты

- **1000$_h$** Device type
  Тип устройства. Модули \slave\can_obj_device.c, \slave__can_devices.c, \slave__can_device_*.h.
- **1001$_h$** Error register
  Регистр ошибок. Модуль \common\can_obj_errors.c.
- **1002$_h$** Manufacturer status register
  Регистр статуса производителя устройства. Модули \slave\can_obj_device.c,

\slave__can_devices.c, \slave__can_device_*.h.

- **1003$_h$** Pre–defined error field
  Список предопределенных ошибок. Модуль \common\can_obj_errors.c.
- **1008$_h$** Manufacturer device name
  Название устройства от производителя. Модули \slave\can_obj_device.c, \slave__can_devices.c, \slave__can_device_*.h.
- **1009$_h$** Manufacturer hardware version
  Версия железа устройства от производителя. Модули \slave\can_obj_device.c, \slave__can_devices.c, \slave__can_device_*.h.
- **100A$_h$** Manufacturer software version
  Версия программного обеспечения устройства от производителя. Модули \slave\can_obj_device.c, \slave__can_devices.c, \slave__can_device_*.h.
- **1010$_h$** Store parameters
  Сохранение значений объектов в энергонезависимой памяти. Модуль \common\can_obj_re_store.c.
- **1011$_h$** Restore default parameters
  Восстановление значений по умолчанию для объектов. Модуль \common\can_obj_re_store.c.
- **1014$_h$** COB–ID EMCY
  COB–ID объекта EMCY. Модуль \common\can_obj_emcy.c.
- **1015$_h$** Inhibit time EMCY
  Время подавления посылок объекта EMCY (кратно 100 мкс). Модуль \common\can_obj_emcy.c.
- **1018$_h$** Identity object
  Объект идентификации устройства. Модули \slave\can_obj_device.c, \slave__can_devices.c, \slave__can_device_*.h.
- **1200$_h$..127F$_h$** SDO server parameter
  SDO параметры сервера. Для оптимизации конечного приложения в библиотеку входят два модуля объектного словаря SDO сервера. В случае использования сервером единственного SDO по умолчанию (как правило) применяется модуль \slave\can_obdsdo_server_default.h. При поддержке сервером от 2 до 128 SDO параметров используется модуль \slave\can_obdsdo_server_num.h. Диспетчер модулей \slave\can_obdsdo_server.c. Фактическое число серверных SDO определяется параметром CAN_NOF_SDO_SERVER. Серверные SDO параметры создаются в объектном словаре каждого узла начиная с индекса 1200$_h$. Обработка операций SDO протокола производится в модуле \common\can_sdo_proc.c.

Параметры режимов и сборки CANopen приложения

Параметры определены в модулях \include__can_defines.h и \include__can_node_id.h.

Важное замечание.
В большинстве случаев модули библиотеки не контролируют значения и диапазоны определения параметров. Поэтому любые их изменения должны производиться исключительно со знанием дела и возможностью справиться с последствиями.

- CAN_APPLICATION_MODE
  Общий режим сборки компилятором конечного приложения:
  MASTER – сборка приложения для master устройства (SDO клиент).
  SLAVE – сборка для slave устройства (SDO сервер).
  TEST – тестовый режим; используется для отладки самой библиотеки. Замыкает приложения тестовых объектов slave и master. Не требует наличия контроллера CAN сети и драйвера канального уровня.
- CAN_NMT_MODE
  Режим сборки конечного приложения для протоколов сетевого менеджера NMT.
  MASTER – устройство поддерживает функциональность NMT master.
  SLAVE – устройство поддерживает функциональность NMT slave.
- CAN_SLAVE_DEVICE_CLASS
  Определяет профиль slave устройства, для которого создается приложение. Предполагается, что программные модули, специфичные для данного устройства и поддерживающие соответствующий профиль, разработаны и имеются в наличии.
- CAN_NETWORK_CONTROLLER
  Номер канала контроллера CAN сети. Значение по умолчанию.
- CAN_BITRATE_INDEX
  Индекс битовой скорости CAN сети. Значение по умолчанию.
- CAN_OS_LINUX, CAN_OS_WIN32
  Определяют тип операционной системы, для которой производится сборка библиотеки.
  CAN_OS_LINUX – Операционная система Linux.
  CAN_OS_WIN32 – Microsoft Windows.
- CAN_ID_MODE
  Длина идентификаторов CAN кадра канального уровня.
  CANID11 – 11-битовый CAN–ID.
  CANID29 – 29-битовый CAN–ID (зарезервирован, в CANopen не используется).
- CAN_FRAME_READ_MODE
  Определяет способ получения CAN кадра канального уровня от драйвера.
  SIGNAL – кадры считываются по сигналу (для операционных систем), либо аппаратному прерыванию CAN контроллера.
  POLL – CAN кадры считываются по опросу из главного цикла программы.
- CAN_BYTE_ORDER
  Порядок следований байт для численных типов данных (целые, реальные). Передача данных по сети начинается с младшего (наименее значимого) байта.
  NORMAL – старший байт расположен по старшему адресу (little-endian).
  REVERSE – старший байт расположен по младшему адресу (big-endian).

- CAN_PDO_MAPPING_MODE
  Задается способ поддержки PDO отображения.
  DYNAMIC – динамически модифицируемое PDO отображение.
  STATIC – статическое (не изменяемое) PDO отображение.
  Динамическое PDO отображение является бит–ориентированным, либо байт–ориентированным. Статическое PDO отображение байт–ориентировано, то есть в одном PDO может содержаться не более восьми объектов, длина каждого из которых кратна 8 битам. Максимальное число объектов приложения, отображаемых динамически в один PDO, может быть определено индивидуально для каждого RPDO и TPDO из диапазона от 1 до 64 с учетом гранулярности. Значение по умолчанию задается параметром CAN_NOF_MAP. Параметр «гранулярность» (granularity) задает минимальное группу бит, которая может быть отображена в динамическое PDO, в диапазоне от 1 (побитовое PDO отображение) до 64 (в PDO может быть отображен один объект длиной 64 бита). Для статического PDO отображения granularity равна нулю. Параметр granularity определен в стандарте электронного описания объектного словаря CANopen устройств (EDS, CiA 306).
- CAN_DYNAMIC_MAPPING_GRANULARITY
  Флаг гранулярности (granularity) динамического PDO отображения.
  MAPBIT – используется бит–ориентированное динамическое PDO отображение (до 64 объектов в одном PDO).
  MAPBYTE – динамическое PDO отображение является байт–ориентированным, то есть в каждом PDO может содержаться не более восьми объектов длина каждого из которых кратна 8 битам.
- CAN_MASTER_RECVCANID_METHOD
  Задает метод, используемый мастером при обработке CAN–ID входящих кадров. (см. «Методы обработки CAN–ID входящих кадров»).
  DYNAMIC – динамический метод обработки.
  STATIC – статический метод обработки.
- CAN_HARD_ACCEPTANCE_FILTER
  Определяет число уровней масочного фильтра входящих кадров CAN контроллера.
  AFSINGLE – одноуровневый фильтр.
  AFDUAL – двухуровневый фильтр.
  AFTRIPLE – трехуровневый фильтр для EN50325-5 (при поддержке драйвером канального уровня CHAI).
- CAN_LED_INDICATOR
  Тип светодиодной индикации состояния устройства (CiA 303 ч. 3)
  COMBINED – используется совмещенный красно/зеленый светодиод.
  SEPARATE – применяются отдельно красный и зеленый светодиоды.
- CAN_CRC_MODE
  Алгоритм подсчета CRC.
  CRCTABLE – табличный байт–оптимизированный подсчет CRC.
  CRCDIRECT – побитовый полиномиальный расчет CRC.
- CAN_OBJECT_EMCY
  Поддержка объекта срочных сообщений EMCY.
  TRUE – EMCY существует и поддерживается.
  FALSE – объекта не существует.
- CAN_OBJECT_TIME
  Поддержка объекта временно*й* метки TIME.
  TRUE – TIME существует и поддерживается.
  FALSE – объекта не существует.

- CAN_OBJECT_RE_STORE
  Поддержка объекта сохранения/восстановления параметров в энергонезависимой памяти.
  TRUE – объект существует и поддерживается.
  FALSE – объекта не существует.
  Объект также поддерживается при активации LSS протокола.
- CAN_OBJECT_ERR_BEHAVIOUR
  Поддержка объекта поведения устройства при возникновении серьезных ошибок.
  TRUE – объект существует и поддерживается.
  FALSE – объекта не существует.
- CAN_PROTOCOL_BLOCK
  Блочный SDO протокол.
  TRUE – протокол поддерживается.
  FALSE – блочный SDO протокол не поддерживается.
- CAN_PROTOCOL_LSS
  Сервис установки уровня (LSS протокол).
  TRUE – LSS протокол активирован.
  FALSE – LSS протокол не поддерживается.
  При активации LSS протокола также активируется объект сохранения/восстановления параметров в энергонезависимой памяти.
- CAN_NOF_NODES
  Полное число узлов CAN сети.
  Master–объекты (SDO, PDO и др.) инициализируются предопределенным распределением CAN идентификаторов в предположении, что узлы сети пронумерованы последовательно. Например, состоящая из трех узлов CAN сеть при инициализации конфигурируется для узлов с номерами 1, 2 и 3. Пользовательское приложение может изменить эту конфигурацию.
- CAN_NOF_RECVCANID_SLAVE
  Максимальное число CAN–ID, которые способен обслуживать slave. Он использует только динамический метод обработки идентификаторов.
- CAN_NOF_RECVCANID_MASTER
  Максимальное число CAN–ID, которые способен обслуживать master при использовании динамического метода обработки идентификаторов.
- CAN_NOF_PREDEF_ERRORS
  Максимальное число регистрируемых ошибок для объекта 1003_h – список предопределенных ошибок.
- CAN_NOF_ERRBEH_SUBIND
  Максимальный субиндекс объекта 1029_h – поведение CAN устройства при возникновении серьезных ошибок.
- CAN_NOF_MAP
  Максимальное число прикладных объектов, которые могут быть динамически отображены в один PDO с учетом гранулярности: от 1 до 64. Для каждого RPDO и TPDO параметр может быть установлен индивидуально в модулях конфигурирования PDO объектов: \common\pdomapping__map_recv_*_*.h для принимаемых PDO и \common\pdomapping__map_tran_*_*.h для передаваемых PDO.
- CAN_NOF_SDO_SERVER
  Число SDO параметров сервера (записей в объектном словаре). Серверные SDO инициализируются в соответствии с предопределенным распределением идентификаторов с учетом номера CAN узла. Пользовательское приложение может изменить начальную конфигурацию.

- CAN_NOF_PDO_RECV_SLAVE
  CAN_NOF_PDO_TRAN_SLAVE
  Число принимаемых RPDO параметров для slave.
  Число передаваемых TPDO параметров для slave.
  Slave PDO инициализируются в соответствии с предопределенным распределением идентификаторов с учетом номера CAN узла. Пользовательское приложение может изменить начальную конфигурацию.
- CAN_NOF_SYNCPDO_MASTER
  Размер каждого FIFO буфера для принимаемых и передаваемых мастером синхронных PDO.
- CAN_TIMERUSEC
  Период CANopen таймера в микросекундах.
  Значение параметра должно быть не менее 100. Период таймера можно изменять в зависимости от требований к разрешению различных временны*х* CANopen объектов: SYNC, SRDO, таймера события PDO, времени подавления PDO и EMCY и т. д.
- CAN_TIMEOUT_RETRIEVE
  Таймаут получения данных из CAN сети для базовой SDO транзакции клиента. Задается в микросекундах. В базовой SDO транзакции клиент ожидает ответ от сервера.
- CAN_TIMEOUT_READ
  Таймаут чтения приложением принятых из CAN сети данных для базовой SDO транзакции клиента. Задается в микросекундах .
- CAN_TIMEOUT_SERVER
  Таймаут базовой SDO транзакции сервера. Задается в микросекундах. В базовой SDO транзакции сервер ожидает запрос очередного сегмента данных от клиента.
- CAN_HBT_PRODUCER_MS
  Значение по умолчанию для периода сердцебиения поставщика в миллисекундах. Инициализирует объект 1017_h (producer heartbeat time).
- CAN_HBT_CONSUMER_MS
  Значение по умолчанию для периода сердцебиения потребителя в миллисекундах. Инициализирует объект 1016_h (consumer heartbeat time) для всех узлов CAN сети.
- CAN_EMCY_INHIBIT_100MCS
  Значение по умолчанию для времени подавления посылок объекта EMCY. Инициализирует объект 1015_h (inhibit time EMCY).
- CAN_RPDO_TRTYPE
  Значение по умолчанию для типа передачи RPDO. Используется для инициализации субиндекса 2 (transmission type) объектов $1400_h..15FF_h$ – коммуникационные параметры принимаемых PDO.
- CAN_TPDO_TRTYPE
  Значение по умолчанию для типа передачи TPDO. Используется для инициализации субиндекса 2 (transmission type) объектов $1800_h..19FF_h$ – коммуникационные параметры передаваемых PDO.
- CAN_TPDO_INHIBIT_100MCS
  Значение по умолчанию для времени подавления посылок TPDO. Инициализирует субиндекс 3 объектов $1800_h..19FF_h$ – коммуникационные параметры передаваемых PDO.
- CAN_RPDO_ET_MS
  Значение по умолчанию для таймера событий RPDO. Инициализирует субиндекс 5 (event timer) объектов $1400_h..15FF_h$ – коммуникационные параметры принимаемых PDO.
- CAN_TPDO_ET_MS
  Значение по умолчанию для таймера событий TPDO. Инициализирует субиндекс 5 (event timer) объектов $1800_h..19FF_h$ – коммуникационные параметры передаваемых PDO.

- CAN_TPDO_SYNC_START
  Значение по умолчанию для начального значения SYNC счетчика TPDO. Инициализирует субиндекс 6 (SYNC start value) объектов $1800_h..19FF_h$ – коммуникационные параметры передаваемых PDO.
- CAN_SIZE_MAXSDOMEM
  Максимальный размер в байтах записи объектного словаря, которая может быть состоятельно передана посредством SDO протокола. Этот параметр используется в сигнало-безопасной функции динамического выделения памяти для определения максимального размера буфера. В случае, если размер объекта превышает CAN_SIZE_MAXSDOMEM, он может быть передан только с использованием без-буферного режима блочного SDO протокола. При использовании блочного протокола передача данных производится, как правило, непосредственно между записями объектного словаря передающей и принимающей сторон. Для этого обеспечивается доступ к соответствующим объектам посредством байтового указателя. Блочный протокол гарантирует состоятельность данных объектного словаря принимающей стороны только после успешного завершения всего цикла обмена.
- CAN_LEN_VISIBLE_STRING
  Максимальная длина типа данных vis–string (видимая строка).
- CAN_NODEID_SLAVE
  Номер CANopen узла, устанавливаемый по умолчанию.
  Допустимые значения 1..127 и 255.
- CAN_SERIAL_NUMBER
  Серийный номер CANopen устройства (объект $1018_h sub4_h$).

API функций master и slave для приложений, которые взаимодействуют с CANopen

int16 pdo_remote_transmit_request(canindex index);

Формирует и посылает удаленный запрос для PDO, заданного коммуникационным параметром **index**. PDO должен быть определен как принимаемый (RPDO), быть действительным и иметь разрешение RTR запроса. Все прикладные объекты, отображенные в соответствующий RPDO, должны быть доступны как по чтению, так и по записи.

Параметры:

- **index** – индекс коммуникационного параметра PDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_COMM_SEND – Коммуникационная ошибка CAN сети: не удалось отправить кадр в сеть.
- CAN_ERRET_NODE_STATE – CAN узел в не операционном состоянии.
- CAN_ERRET_OBD_NOOBJECT – Ошибка конфигурирования PDO: объекта с индексом **index** не существует либо он не является RPDO.
- CAN_ERRET_PDO_INVALID – PDO в не действительном состоянии.
- CAN_ERRET_PDO_NORTR – Для данного PDO удаленный запрос запрещен.

int16 transmit_can_pdo(canindex index);

Формирует и посылает TPDO, определяемый коммуникационным параметром **index**. Обслуживает TPDO со следующими типами передачи:

- 0 – ациклические синхронные;
- 254, 255 – асинхронные;

PDO должен быть определен как передаваемый (TPDO), быть действительным и не находиться в состоянии подавления.

Параметры:

- **index** – индекс коммуникационного параметра TPDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_COMM_SEND – Коммуникационная ошибка CAN сети: не удалось отправить кадр в сеть.
- CAN_ERRET_NODE_STATE – CAN узел в не операционном состоянии.
- CAN_ERRET_OBD_NOOBJECT – Ошибка конфигурирования PDO: объекта с индексом **index** не существует либо он не является TPDO. Теоретически возможная ошибка: при формирования PDO отображения был определен несуществующий объект.
- CAN_ERRET_OBD_NOSUBIND – Теоретически возможная ошибка: при формирования PDO отображения был определен несуществующий субиндекс объекта.
- CAN_ERRET_PDO_ERRMAP – В отображении PDO указан неверный размер объекта, либо полная длина отображаемых объектов превышает максимальный размер PDO (64 бита).
- CAN_ERRET_PDO_INHIBIT – PDO находится в состоянии подавления.
- CAN_ERRET_PDO_INVALID – PDO находится в не действительном состоянии.
- CAN_ERRET_PDO_MAP_DEACT – PDO отображение деактивировано.
- CAN_ERRET_PDO_TRTYPE – неверный тип передачи PDO.

void produce_time(unsigned32 ms, unsigned16 days);

Формирует и посылает в сеть объект временной метки TIME. Объект длиной шесть байт формируется в соответствии с описанием структуры TIME_OF_DAY стандарта

CiA 301. Для параметра **ms** используется 28 младших бит.
Параметры:
- **ms** – время в миллисекундах после ноля часов.
- **days** – число дней, прошедших с 01 января 1984 г.

API функций master для приложений, которые взаимодействуют с CANopen

void can_sdo_client_transfer(struct sdocltappl *ca);

Выполняет полную SDO транзакцию передачи данных между клиентом и сервером. Режимы проведения транзакции, ее условия и результаты содержатся в структуре ***ca**.

Параметры:

- **ca.operation** – определяет базовый режим передачи SDO. Обязательно задается пользователем и модифицируется функцией. CAN_SDOPER_DOWNLOAD – download или CAN_SDOPER_UPLOAD – upload. Если размер данных не превышает 4 байта, используется ускоренный режим передачи. При размере данных более 4 байт, но не превышающем CAN_SIZE_MAXSDOMEM, применяется сегментированный режим. При бо́льшем размере данных используется блочный SDO протокол. После выполнения функции параметр **ca.operation** содержит код режима, фактически использованного при SDO обмене:
  CAN_SDOPER_(UP/DOWN)_EXPEDITED – ускоренный,
  CAN_SDOPER_(UP/DOWN)_SEGMENTED – сегментированный,
  CAN_SDOPER_(UP/DOWN)_BLOCK – блочный режим.
  Сам базовый режим (UPload или DOWNload) всегда остается неизменным.
- **ca.node** – номер узла SDO сервера. Определяется функцией самостоятельно и не задается пользователем. Содержит номер CAN узла slave устройства для осуществления доступа к отображенному в мастере прикладному профилю этого устройства. Извлекается функцией из коммуникационного SDO параметра клиента (объекты 1280_h..$12FF_h$).
- **ca.datasize** – размер данных в байтах. Задается пользователем либо определяется функцией. Содержит размер передаваемых посредством SDO данных в байтах. Должен быть задан, когда указатель **ca.datapnt** не равен NULL и в этом случае не изменяется функцией. При значении указателя **ca.datapnt** равном NULL, **ca.datasize** определяется функцией самостоятельно.
- **ca.datapnt** – указатель на локальный буфер. Может задаваться пользователем. Если указатель не задан (равен NULL) и используется блочный SDO протокол, то определяется функцией, в противном случае не изменяется. Если указатель **ca.datapnt** задан (не равен NULL), то передаваемые SDO данные будут считываться или записываться в локальный буфер, определяемый указателем. В этом случае обязательно должен быть определен размер данных **ca.datasize**. Когда указатель **ca.datapnt** не задан (равен NULL), используется запись мастер–отображения объектного словаря. Она определяется параметрами **ca.node** (номер CAN узла SDO сервера / slave устройства), **ca.si.index** (индекс прикладного объекта) и **ca.si.subind** (субиндекс прикладного объекта).
- **sdoixs** – структура SDO индексов. Обязательно задается пользователем и не модифицируется функцией. **ca.si.sdoind** индекс коммуникационного SDO параметра клиента (1280_h .. $12FF_h$). **ca.si.index** и **ca.si.subind** соответственно индекс и субиндекс прикладного объекта, передаваемого с помощью SDO.
- **sdostatus** – структура статуса транзакции. Устанавливается только функцией и содержит код завершения (статус) SDO транзакции клиента. **ca.ss.state** – статус завершения SDO транзакции. **ca.ss.abortcode** – аборт код в соответствии со стандартом CiA 301; устанавливается при статусе завершения SDO транзакции CAN_TRANSTATE_SDO_SRVABORT. Если транзакция выполнена успешно, статус завершения равен CAN_TRANSTATE_OK. В противном случае устанавливается одно из значений кода ошибки:
  CAN_TRANSTATE_OBD_ZERO – запись объектного словаря имеет нулевой размер;
  CAN_TRANSTATE_OBD_READ – ошибка чтения объектного словаря;

CAN_TRANSTATE_OBD_WRITE – ошибка записи объектного словаря;
CAN_TRANSTATE_OBD_NOOBJECT – нет такого объекта (индекса) в словаре;
CAN_TRANSTATE_OBD_NOSUBIND – нет такого субиндекса;
CAN_TRANSTATE_OBD_MALLOC – ошибка выделения динамического буфера;
CAN_TRANSTATE_SDO_RETRANSMIT – превышено число повторных передач сегмента данных (блочный протокол);
CAN_TRANSTATE_SDO_BLKSIZE – неверное число сегментов в блоке данных (блочный протокол);
CAN_TRANSTATE_SDO_SEQNO – неверный номер сегмента (блочный протокол);
CAN_TRANSTATE_SDO_CRC – ошибка CRC (блочный протокол);
CAN_TRANSTATE_SDO_SUB – неверная субкоманда (блочный протокол);
CAN_TRANSTATE_SDO_TOGGLE – ошибка мерцающего бита (toggle) в протоколе сегментированной передачи;
CAN_TRANSTATE_SDO_DATASIZE – неверный размер данных в сегменте;
CAN_TRANSTATE_SDO_OBJSIZE – размеры объекта, известные клиенту и серверу, не совпадают;
CAN_TRANSTATE_SDO_MODE – несоответствие режимов передачи клиента и сервера (SDO upload протокол)
CAN_TRANSTATE_SDO_MPX – несоответствие мультиплексоров клиента и сервера;
CAN_TRANSTATE_SDO_SRVABORT – от сервера получен аборт SDO протокола;
CAN_TRANSTATE_SDO_INVALID – SDO не действительно;
CAN_TRANSTATE_SDO_WRITERR – ошибка передачи SDO в CAN сеть;
CAN_TRANSTATE_SDO_SCSERR – SDO клиент получил от сервера неверную или не известную команду;
CAN_TRANSTATE_SDO_TRANS_TIMEOUT – внутренний таймаут базовой транзакции SDO клиента;
CAN_TRANSTATE_SDO_NET_TIMEOUT – сетевой таймаут базовой транзакции SDO клиента;
CAN_TRANSTATE_SDO_READ_TIMEOUT – таймаут чтения данных приложением; базовая SDO транзакция клиента сброшена;
CAN_TRANSTATE_SDO_NOWORKB – переполнение рабочего буфера базовых SDO транзакций клиента;
CAN_TRANSTATE_SDO_NODE – ошибка чтения номера узла SDO сервера;
CAN_TRANSTATE_ERROR – общая ошибка.

int16 get_pdo_node(canindex index, cannode *node);

Чтение номера CAN узла для PDO.

Используется в мастер-приложении.

Параметры:

- ***node** – значение номера узла для коммуникационного параметра PDO **index**.
- **index** – индекс коммуникационного параметра PDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – PDO объекта нет в словаре.

int16 put_pdo_node(canindex index, cannode node);

Запись номера CAN узла для PDO.

Используется в мастер-приложении. Номер CAN узла для PDO задается отдельной функцией, поскольку его значение не предусмотрено в соответствующих записях объектного словаря.

Параметры:

- **node** – значение номера узла для коммуникационного параметра PDO **index**.

- **index** – индекс коммуникационного параметра PDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – PDO объекта нет в словаре.

void nmt_master_command(unsigned8 cs, cannode node);

Формирует и посылает в CAN сеть NMT кадр, содержащий NMT команду **cs** и номер узла **node**. Функция не осуществляет каких–либо проверок значений NMT команды и номера CAN узла.

Параметры:

- **cs** – NMT команда.
- **node** – номер CAN узла.

canbyte *node_get_manstan_objpointer(cannode node, canindex index, cansubind subind);

Получение байтового указателя на объект словаря мастер–отображения.

Параметры:

- **node** – номер slave узла.
- **index** – индекс отображаемого прикладного объекта.
- **subind** – субиндекс отображаемого прикладного объекта.

Возвращаемые значения:

- не равно NULL – байтовый указатель на объект, определяемый аргументами функции.
- NULL – соответствующий объект не доступен посредством указателя.

int16 node_see_manstan_access(cannode node, canindex index, cansubind subind);

Определяет маску доступа к записи объектного словаря мастер–отображения.

Параметры:

- **node** – номер slave узла.
- **index** – индекс отображаемого прикладного объекта.
- **subind** – субиндекс отображаемого прикладного объекта.

Возвращаемые значения: маска доступа > 0 (биты 0..14); ошибка <= 0 (установлен бит 15).

- CAN_MASK_ACCESS_PDO – флаг допустимости PDO отображения объекта (бит_0 = 1).
- CAN_MASK_ACCESS_RO – флаг доступа к объекту по чтению (бит_1 = 1).
- CAN_MASK_ACCESS_WO – флаг доступа к объекту по записи (бит_2 = 1).
- CAN_MASK_ACCESS_RW – доступ к объекту по чтению и записи (бит_1 = 1 и бит_2 = 1).
- = 0 – объект нулевого размера, нет доступа.
- CAN_ERRET_OBD_INVNODE – неверное значение номера CAN узла.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int32 node_get_manstan_objsize(cannode node, canindex index, cansubind subind, int16 unit);

Запрос размера объекта из словаря мастер–отображения. Эта функция также определяет наличие соответствующего объекта в словаре. Размер может быть представлен в байтах (параметр **unit** = BYTES) или в битах (параметр **unit** = BITS). Размер в битах используется для бит–ориентированного PDO отображения.

Параметры:

- **node** – номер slave узла.
- **index** – индекс отображаемого прикладного объекта.
- **subind** – субиндекс отображаемого прикладного объекта.
- **unit** – единица измерения размера объекта: байт (BYTES) или бит (BITS).

Возвращаемые значения: размер объекта > 0; ошибка < 0.
- > 0 – размер объекта в единицах **unit**.
- CAN_ERRET_OBD_INVNODE – неверное значение номера CAN узла.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int16 node_get_manstan_objtype(cannode node, canindex index, cansubind subind);

Запрос типа объекта из словаря мастер–отображения.

Параметры:
- **node** – номер slave узла.
- **index** – индекс отображаемого прикладного объекта.
- **subind** – субиндекс отображаемого прикладного объекта.

Возвращаемые значения: тип объекта > 0; ошибка < 0.
- > 0 – индекс типа объекта (0001_h..$001F_h$: статические типы данных объектного словаря).
- CAN_ERRET_OBD_INVNODE – неверное значение номера CAN узла.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int16 node_read_manstan_objdict(cannode node, canindex index, cansubind subind, canbyte *data);

Чтение объекта из словаря мастер–отображения. Результат преобразуется в байтовый формат и размещается по адресу ***data**. Порядок следования байт не изменяется. Приложение должно выделить буфер, достаточный для размещения всего объекта. Размер объекта при необходимости может быть определен с помощью функции node_get_manstan_objsize(...).

Параметры:
- **node** – номер slave узла.
- **index** – индекс отображаемого прикладного объекта.
- **subind** – субиндекс отображаемого прикладного объекта.
- ***data** – байтовый указатель на размещаемые данные.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_INVNODE – неверное значение номера CAN узла.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_WRITEONLY – попытка чтения только записываемого объекта.

int16 node_read_manstan_objdict_network(cannode node, canindex index, cansubind subind, canbyte *data);

Чтение объекта из словаря мастер–отображения с приведением порядка следования байт к необходимому для передачи данных по сети. Если CAN_BYTE_ORDER = NORMAL функция полностью аналогична node_read_manstan_objdict(...). При CAN_BYTE_ORDER = REVERSE после чтения объекта из словаря для основных численных типов данных порядок следования байт инвертируется.

Параметры:

см. node_read_manstan_objdict(...).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. node_read_manstan_objdict(...).

int16 node_write_manstan_objdict(cannode node, canindex index, cansubind subind, canbyte *data);

Запись объекта в словарь мастер–отображения. Функция помещает объект,

расположенный по адресу ***data**, в запись объектного словаря. Порядок следования байт не изменяется. До вызова функции приложение должно привести соответствующие данные к байтовому виду.

Параметры:

- **node** – номер slave узла.
- **index** – индекс отображаемого прикладного объекта.
- **subind** – субиндекс отображаемого прикладного объекта.
- ***data** – байтовый указатель на размещенные данные.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_INVNODE – неверное значение номера CAN узла.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_READONLY – попытка записи только читаемого объекта.

int16 node_write_manstan_objdict_network(cannode node, canindex index, cansubind subind, canbyte *data);

Запись объекта в словарь мастер–отображения с приведением порядка следования байт к необходимому для записи в приложение. Если CAN_BYTE_ORDER = NORMAL функция полностью аналогична node_write_manstan_objdict(...). При CAN_BYTE_ORDER = REVERSE до записи объекта в словарь для основных численных типов данных порядок следования байт инвертируется.

Параметры:

см. node_write_manstan_objdict(...).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. node_write_manstan_objdict(...).

int16 client_see_access(canindex index, cansubind subind);

Определяет маску доступа к записи объектного словаря для коммуникаций клиента .

Параметры:

- **index** – индекс коммуникационного объекта.
- **subind** – субиндекс коммуникационного объекта.

Возвращаемые значения: маска доступа > 0 (биты 0..14); ошибка <= 0 (установлен бит 15).

- CAN_MASK_ACCESS_PDO – флаг допустимости PDO отображения объекта (бит_0 = 1).
- CAN_MASK_ACCESS_RO – флаг доступа к объекту по чтению (бит_1 = 1).
- CAN_MASK_ACCESS_WO – флаг доступа к объекту по записи (бит_2 = 1).
- CAN_MASK_ACCESS_RW – доступ к объекту по чтению и записи (бит_1 = 1 и бит_2 = 1).
- = 0 – объект нулевого размера, нет доступа.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int32 client_get_object_size(canindex index, cansubind subind, int16 unit);

Запрос размера объекта из объектного словаря для коммуникаций клиента. Эта функция также определяет наличие соответствующего объекта в словаре. Размер может быть представлен в байтах (параметр **unit** = BYTES) или в битах (параметр **unit** = BITS). Размер в битах используется для бит–ориентированного PDO отображения.

Параметры:

- **index** – индекс коммуникационного объекта.
- **subind** – субиндекс коммуникационного объекта.
- **unit** – единица измерения размера объекта: байт (BYTES) или бит (BITS).

Возвращаемые значения: размер объекта > 0; ошибка < 0.
- > 0 – размер объекта в единицах **unit**.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int16 client_read_object_dictionary(canindex index, cansubind subind, canbyte *data);

Чтение объекта из объектного словаря для коммуникаций клиента. Результат конвертируется в байтовый формат и размещается по адресу ***data**. Порядок следования байт не изменяется. Приложение должно выделить буфер, достаточный для размещения всего объекта. Размер объекта при необходимости может быть определен с помощью функции client_get_object_size(...).

Параметры:
- **index** – индекс коммуникационного объекта.
- **subind** – субиндекс коммуникационного объекта.
- ***data** – байтовый указатель на размещаемые данные.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_WRITEONLY – попытка чтения только записываемого объекта.

int16 client_write_object_dictionary(canindex index, cansubind subind, canbyte *data);

Запись объекта в объектный словаря для коммуникаций клиента. Функция помещает объект, расположенный по адресу ***data**, в запись словаря для коммуникаций. Порядок следования байт не изменяется. До вызова функции приложение должно привести соответствующие данные к байтовому виду.

Параметры:
- **index** – индекс коммуникационного объекта.
- **subind** – субиндекс коммуникационного объекта.
- ***data** – байтовый указатель на размещенные данные.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_READONLY – попытка записи только читаемого объекта.
- CAN_ERRET_OBD_VALRANGE – ошибка диапазона записываемого значения.
- CAN_ERRET_OBD_OBJACCESS – в текущем состоянии объект не может быть изменен.
- CAN_ERRET_OBD_PARINCOMP – несовместимость записываемого значения объекта.

int16 client_read_obd_u32(cannode node, canindex index, cansubind subind, unsigned32 *du32);

Чтение объекта размером до 32 бит. Функция служит для облегчения доступа к объектам, длина которых не превышает 32 бита. Она читает объект, определяемый параметрами **node** – номер CAN узла, **index** – индекс и **subind** – субиндекс. Результат размещается в параметре ***du32**. Если **node** = 0, читаются данные из объектного словаря для коммуникаций клиента, иначе из словаря мастер–отображения. Размер объекта не должен превышать 32 бита, в противном случае поведение функции и результат не предсказуемы.

Параметры:

см. client_read_object_dictionary(...), node_read_manstan_objdict(...).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. client_read_object_dictionary(...), node_read_manstan_objdict(...).

int16 client_write_obd_u32(cannode node, canindex index, cansubind subind, unsigned32 du32);

Запись объекта размером до 32 бит. Функция служит для облегчения доступа к объектам, длина которых не превышает 32 бита. Она помещает значение **du32** в запись словаря, определяемую параметрами **node** – номер CAN узла, **index** – индекс и **subind** – субиндекс объекта. Если параметр **node** = 0, данные записываются в объектный словарь для коммуникаций клиента, иначе в словарь мастер–отображения. Размер объекта не должен превышать 32 бита, в противном случае поведение функции и результат не предсказуемы.

Параметры:

см. client_write_object_dictionary(...), node_write_manstan_objdict(...).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. client_write_object_dictionary(...), node_write_manstan_objdict(...).

int16 produce_emcy_default(unsigned16 errorcode);

Регистрирует в мастере EMCY с кодом ошибки **errorcode**. Срочное сообщение не передается в CAN сеть.

Параметры:

- **errorcode** – код ошибки.

Возвращаемые значения:

- CAN_RETOK – нормальное завершение.

API функций slave для приложений, которые взаимодействуют с CANopen

canbyte *server_get_object_pointer(canindex index, cansubind subind);

Возвращает байтовый указатель на объект словаря slave устройства.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.

Возвращаемые значения:

- не равно NULL – байтовый указатель на объект, определяемый аргументами функции.
- NULL – соответствующий объект не доступен посредством указателя.

int16 server_see_access(canindex index, cansubind subind);

Определяет маску доступа к записи объектного словаря slave устройства.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.

Возвращаемые значения: маска доступа > 0 (биты 0..14); ошибка <= 0 (установлен бит 15).

- CAN_MASK_ACCESS_PDO – флаг допустимости PDO отображения объекта (бит_0 = 1).
- CAN_MASK_ACCESS_RO – флаг доступа к объекту по чтению (бит_1 = 1).
- CAN_MASK_ACCESS_WO – флаг доступа к объекту по записи (бит_2 = 1).
- CAN_MASK_ACCESS_RW – доступ к объекту по чтению и записи (бит_1 = 1 и бит_2 = 1).
- = 0 – объект нулевого размера, нет доступа.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int32 server_get_object_size(canindex index, cansubind subind, int16 unit);

Запрос размера объекта из словаря slave устройства. Эта функция также определяет наличие соответствующего объекта в словаре. Размер может быть представлен в байтах (параметр **unit** = BYTES) или в битах (параметр **unit** = BITS). Размер в битах используется для бит–ориентированного PDO отображения.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- **unit** – единица измерения размера объекта: байт (BYTES) или бит (BITS).

Возвращаемые значения: размер объекта > 0; ошибка < 0.

- > 0 – размер объекта в единицах **unit**.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int16 server_get_object_type(canindex index, cansubind subind);

Запрос типа объекта из словаря slave устройства.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.

Возвращаемые значения: тип объекта > 0; ошибка < 0.

- > 0 – индекс типа объекта (0001_h..$001F_h$: статические типы данных объектного словаря).
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int16 server_read_object_dictionary(canindex index, cansubind subind, canbyte *data);

Чтение объекта из словаря slave устройства. Результат конвертируется в байтовый вид и размещается по адресу ***data**. Порядок следования байт не изменяется. Приложение должно выделить буфер, достаточный для размещения всего объекта. Размер объекта при необходимости может быть определен с помощью функции server_get_object_size(...).

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- ***data** – байтовый указатель на размещаемые данные.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_WRITEONLY – попытка чтения только записываемого объекта.

int16 server_read_obd_network(canindex index, cansubind subind, canbyte *data);

Чтение объекта из словаря slave устройства с приведением порядка следования байт к необходимому для передачи данных по сети. Если CAN_BYTE_ORDER = NORMAL функция полностью аналогична server_read_object_dictionary(...). При CAN_BYTE_ORDER = REVERSE после чтения объекта из словаря для основных численных типов данных порядок следования байт инвертируется.

Параметры:

см. server_read_object_dictionary(...).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. server_read_object_dictionary(...).

int16 server_write_object_dictionary(canindex index, cansubind subind, canbyte *data);

Запись объекта в словарь slave устройства. Функция помещает объект, расположенный по адресу ***data**, в запись объектного словаря. Порядок следования байт не изменяется. До вызова функции приложение должно привести соответствующие данные к байтовому виду.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- ***data** – байтовый указатель на размещенные данные.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_READONLY – попытка записи только читаемого объекта.

int16 server_write_obd_network(canindex index, cansubind subind, canbyte *data);

Запись объекта в словарь slave устройства с приведением порядка следования байт к необходимому для записи. Если CAN_BYTE_ORDER = NORMAL функция полностью аналогична server_write_object_dictionary(...). При CAN_BYTE_ORDER = REVERSE до записи объекта в словарь для основных численных типов данных порядок следования байт инвертируется.

Параметры:

см. server_write_object_dictionary(...).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. server_write_object_dictionary(...).

int16 server_read_obd_u32(canindex index, cansubind subind, unsigned32 *du32);

Чтение объекта slave устройства размером до 32 бит. Функция служит для облегчения доступа к объектам, длина которых не превышает 32 бита. Она читает объект, определяемый параметрами **index** – индекс и **subind** – субиндекс. Результат размещается в параметре ***du3**2. Размер объекта не должен превышать 32 бита, в противном случае поведение функции и результат не предсказуемы.

Параметры:

см. server_read_object_dictionary(...).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. server_read_object_dictionary(...).

int16 server_write_obd_u32(canindex index, cansubind subind, unsigned32 du32);

Запись объекта slave устройства размером до 32 бит. Функция служит для облегчения доступа к объектам, длина которых не превышает 32 бита. Она помещает значение **du32** в запись словаря, определяемую параметрами **index** – индекс и **subind** – субиндекс объекта. Размер объекта не должен превышать 32 бита, в противном случае поведение функции и результат не предсказуемы.

Параметры:

см. server_write_object_dictionary(...).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. server_write_object_dictionary(...).

int16 produce_emcy(unsigned16 errorcode, unsigned16 addinf, canbyte *mserr);

Создает объект EMCY с полной информацией об ошибке. Заносит а список предопределенных ошибок (объект 1003_h) код ошибки **errorcode** совместно с дополнительной информацией **addinf**. Затем формирует и отсылает EMCY с кодом **errorcode**, текущим состоянием регистра ошибок и полем ошибки производителя устройства ***mserr** (используются первые пять байт). EMCY должен быть действительным, а время подавления его посылок должно истечь.

Параметры:

- **errorcode** – код ошибки EMCY.
- **addinf** – дополнительная информация об ошибке.
- ***mserr** – поле ошибки производителя устройства (5 байт).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_EMCY_INVALID – объект EMCY не действителен.
- CAN_ERRET_EMCY_INHIBIT – объект EMCY находится в состоянии подавления.
- CAN_ERRET_NODE_STATE – CAN узел находится в состоянии останова или инициализации.
- CAN_ERRET_COMM_SEND – Коммуникационная ошибка CAN сети: не удалось отправить кадр в сеть.

int16 produce_emcy_default(unsigned16 errorcode);

Создает объект EMCY с минимальной информацией об ошибке. Используется только код ошибки **errorcode**. Дополнительная информация в списке предопределенных ошибок и поле ошибки производителя устройства отсутствует (сбрасываются в ноль).

Параметры:

- **errorcode** – код ошибки EMCY.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. produce_emcy(...).

void clear_error_register(unsigned8 mask);

Производит побитовую очистку регистра ошибок (объект 1001_h). Нулевой бит регистра (общая ошибка) сбрасывается лишь при условии очистки всех остальных бит. При этом выдается сообщение EMCY с нулевым значением кода ошибки (сброс ошибки). Коды ошибок в диапазоне $1000_h..10FF_h$ устанавливают только бит общей ошибки, который, в отсутствии других ошибок, будет сброшен при любом значении **mask**.

Параметры:

- **mask** – битовая маска. Очищаются биты, для которых в маске установлено значение 1.

int16 get_flash_nodeid();

Чтение значения номера CAN узла из энергонезависимой памяти.

Возвращаемые значения: номер CAN узла >= 0; ошибка < 0.

- CAN_ERRET_FLASH_DATA – Данные в энергонезависимой памяти ошибочны или не состоятельны.
- CAN_ERRET_FLASH_VALUE – Значение параметра не записано в энергонезависимую память.

int16 get_flash_bitrate_index();

Чтение значения индекса битовой скорости CAN сети из энергонезависимой памяти.

Возвращаемые значения: индекс скорости >= 0; ошибка < 0.

см. get_flash_nodeid().

int16 put_flash_nodeid(cannode node);

Сохранение номера CAN узла в энергонезависимой памяти.

Параметры:

- **node** – сохраняемый номер CAN узла.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_FLASH_INIT – Ошибка инициализации (очистки) страницы энергонезависимой памяти.
- CAN_ERRET_FLASH_DATA – Данные, записанные в энергонезависимую память, ошибочны или не состоятельны.

int16 put_flash_bitrate_index(unsigned8 br);

Сохранение индекса битовой скорости CAN сети в энергонезависимой памяти.

Параметры:

- **br** – сохраняемый индекс битовой скорости.

Возвращаемые значения:

см. put_flash_nodeid(...).

API функций, редактируемых пользователем

Полное программирование этих функций осуществляется в зависимости от требований конечного приложения.

unsigned32 read_dev_type_object(canindex index, cansubind subind);

Задает описание типа устройства, регистра статуса производителя и объекта идентификации (identity object). Размещается в модулях \slave__can_device_*.h для различных устройств. Вызывается при чтении соответствующих индексов объектного словаря.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.

Возвращаемое значение:

- Значение соответствующего объекта.

void read_dev_string_object(canindex index, cansubind subind, canbyte *data);

Задает символьные описания устройства: его имя, версии железа и программного обеспечения. Размещается в модулях \slave__can_device_*.h для различных устройств. Вызывается при чтении соответствующих индексов объектного словаря.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- ***data** – байтовый указатель на данные типа vis–string, которые являются символьным описанием устройства.

cannode get_node_id(void);

Возвращает номер узла CANopen устройства, модуль \common__can_init.c. Для master приложения функция должна возвращать ноль.

Возвращаемое значение:

- Номер узла CAN устройства (1..127 и 255 для slave устройств). Считывается из энергонезависимой памяти, либо задается, например, переключателями.

unsigned8 get_bit_rate_index(void);

Возвращает индекс битовой скорости CAN сети, модуль \common__can_init.c.

Возвращаемое значение:

- Индекс битовой скорости CAN сети. Считывается из энергонезависимой памяти, либо задается, например, переключателями.

unsigned32 get_serial_number(void);

Возвращает серийный номер CANopen slave устройства, модуль \common__can_init.c.

Возвращаемое значение:

- Серийный номер slave устройства (объект $1018_h sub4_h$).

void consume_sync(unsigned8 sc);

Обрабатывает объект синхронизации SYNC. Размещается в модуле \common__can_events.c. Используется в master и slave. Вызывается при получении объекта синхронизации.

Параметры:

- **sc** – текущее значение SYNC счетчика (диапазон от 1 до 240).

void no_sync_event(void);

Потребитель SYNC не получил объекта синхронизации в течение промежутка времени, заданного объектом 1006_h (период объекта синхронизации в микросекундах). Размещается в модуле \common__can_events.c. Используется в master и slave. Предусматривает как минимум соответствующую светодиодную индикацию.

void consume_time(canframe *cf);

Обработка объекта временно*й* метки TIME. Размещается в модуле \common__can_events.c Используется в master и slave. Вызывается при получении объекта временно*й* метки и может использоваться для коррекции локального времени устройства.
Параметры:

- ***cf** – CAN кадр, содержащий объект временно*й* метки в формате структуры TIME_OF_DAY.

void consume_controller_error(canev ev);

Обработчик сигнала ошибок от CAN контроллера. Размещается в модуле \common__can_events.c. Используется в master и slave. Ошибка bus off отключения узла от CAN шины обрабатывается согласно настройкам объекта 1029_h – поведение устройства при возникновении серьезных ошибок. Обработка остальных ошибок предусматривает передачу соответствующего EMCY и светодиодную индикацию. Коды ошибок определены в заголовочном файле CAN драйвера канального уровня.
Параметры:

- **ev** (тип int16) – код ошибки:
  CIEV_BOFF – bus off,
  CIEV_EWL – error warning limit,
  CIEV_HOVR – hardware overrun,
  CIEV_SOVR – software overrun.
  CIEV_WTOUT – write timeout occurred,

void pdo_activated_master(cannode node, canindex index, cansubind subind);

Сообщает об активировании PDO в master. Размещается в модуле \common__can_events.c. Предназначена для информирования приложений о том, что отображенный в PDO объект был успешно записан в объектный словарь мастер–отображения устройства.
Параметры:

- **node** – номер slave узла.
- **index** – индекс прикладного объекта.
- **subind** – субиндекс прикладного объекта.

void pdo_activated_slave(canindex index, cansubind subind);

Сообщает об активировании PDO в slave. Размещается в модуле \common__can_events.c. Предназначена для информирования приложений о том, что отображенный в PDO объект был успешно записан в объектный словарь slave устройства.
Параметры:

- **index** – индекс прикладного объекта.
- **subind** – субиндекс прикладного объекта.

void master_emcy(unsigned16 errorcode);

Возникновение EMCY в мастере. Размещается в модуле \common__can_events.c. Вызывается, когда в мастере возникает событие с кодом **errorcode**. При этом срочное сообщение не передается в CAN сеть.

Параметры:
- **errorcode** – код ошибки.

void consume_emcy(canframe *cf);

Обрабатывает объект EMCY. Размещается в модуле \common__can_events.c. Вызывается при получении EMCY сообщения от какого–либо slave устройства. Используется только в мастере.
Параметры:
- ***cf** – CAN кадр EMCY.

void can_client_state(struct sdocltappl *ca);

Информирование о статусе SDO транзакции клиента. Размещается в модуле \common__can_events.c. Сообщает о статусе SDO транзакции клиента после ее завершения. Используется только в мастере.
Параметры:
- ***ca** – структура для взаимодействия с приложением клиента при обмене данными с помощью SDO протокола.

void heartbeat_event(cannode node);

Обрабатывает событие сердцебиения (heartbeat event) со статусом ”occurred”. Размещается в модуле \common__can_events.c. Вызывается при отсутствие сердцебиения для узла **node**. Используется только в NMT–master.
Параметры:
- **node** – номер узла NMT slave.

void heartbeat_resolved(cannode node);

Обрабатывает событие сердцебиения (heartbeat event) со статусом ”resolved”. Размещается в модуле \common__can_events.c. Вызывается при возобновлении поступления посылок сердцебиения для узла **node**. Используется только в NMT–master.
Параметры:
- **node** – номер узла NMT slave.

void node_guarding_event(cannode node);

Обрабатывает событие node guarding event со статусом ”occurred”. Размещается в модуле \common__can_events.c. Вызывается при отсутствии подтверждения в протоколе охраны узла для узла **node**. Используется только в NMT–master.
Параметры:
- **node** – номер узла NMT slave.

void node_guarding_resolved(cannode node);

Обрабатывает событие node guarding event со статусом ”resolved”. Размещается в модуле \common__can_events.c. Вызывается при возобновлении подтверждений от узла **node** в протоколе охраны узла. Используется только в NMT–master.
Параметры:
- **node** – номер узла NMT slave.

void bootup_event(cannode node);

Обрабатывает событие загрузки узла (bootup event). Размещается в модуле \common__can_events.c. Вызывается при возникновении события загрузки для узла **node**. Используется только в NMT–master.
Параметры:

- **node** – номер узла NMT slave.

void node_state_event(cannode node, canbyte state);

Регистрирует NMT состояния узла **node**, полученное с использованием протокола сердцебиения либо охраны узла. Размещается в модуле \common__can_events.c. Вызывается при каждом получении состояния любого NMT slave узла. Используется только в NMT–master.

Параметры:

- **node** – номер узла NMT slave.
- **state** – NMT состояние узла **node**.

void life_guarding_event(void);

Обрабатывает событие life guarding event со статусом ”occurred”. Размещается в модуле \common__can_events.c. Вызывается при отсутствии запросов в протоколе охраны узла. Используется только в NMT–slave.

void life_guarding_resolved(void);

Обрабатывает событие life guarding event со статусом ”resolved”. Размещается в модуле \common__can_events.c. Вызывается при возобновлении запросов в протоколе охраны узла. Используется только в NMT–slave.

void no_pdo_event(canindex index);

Не получено RPDO до истечения его таймера события. Размещается в модуле \common__can_events.c. Используется в master и slave. Предусматривает как минимум соответствующую светодиодную индикацию и передачу EMCY.

Параметры:

- **index** – индекс коммуникационного объекта RPDO.

void can_timer_overlap(void);

Зарегистрировано наложение тиков CANopen таймера. Размещается в модуле \common__can_events.c. Используется в master и slave. Предусматривает как минимум передачу соответствующего EMCY.

void can_cache_overflow(canbyte state);

Переполнен выходной CANopen кэш. Размещается в модуле \common__can_events.c. Используется в master и slave. Предусматривает как минимум регистрацию в объекте ошибок.

Параметры:

- **state** – NMT состояние узла.

void can_init_pdo_map(void);

Инициализация статических PDO отображений. Редактируемые компоненты размещаются в модуле \common\pdomapping__map__static.h. Используется только в slave, когда параметр CAN_PDO_MAPPING_MODE = STATIC. Задает отображение всех принимаемых и передаваемых PDO при использовании статического метода отображения.

API функций общего управления

void can_set_datalink_layer(unsigned8 mode);

Функция управления логическим доступом к CAN сети (CAN драйверу). Осуществляет подключение и отключение канального уровня CAN по записи. Попытки вывода данных в физически отсоединенную CAN сеть могут приводить к значительным задержкам вследствие возникновения таймаутов в драйвере, а при переполнении кэша – и в самой CANopen библиотеке. NMT slave устройство логически вновь подключается к CAN сети при получении любой адресованной ему NMT команды.

Параметры:

- **mode** – режим логического доступа к канальному уровню CAN сети (CAN драйверу) по записи. ON – штатный режим работы: все передаваемые кадры отправляются в CAN сеть. OFF – все кадры, как ожидающие передачи, так и направляемые в CAN сеть аннулируются. При инициализации библиотеки устанавливается штатный режим с отправкой всех кадров в сеть.

API системно–зависимых функций

Эти функции размещаются в соответствующих модулях «корневой» директории CANopen. Модуль __can_system.c служит диспетчером подключения соответствующего системно–зависимого модуля при сборке приложения.

void can_sleep(int32 microseconds);

Функция временн*о*й задержки.

Параметры:

- **microseconds** – временн*а*я задержка в микросекундах. Точное время задержки определяется разрешением соответствующего таймера системы. Любое положительное значение аргумента функции должно обеспечивать отличную от нуля задержку.

void can_init_system_timer(void (*handler)(void));

Инициализация CANopen таймера. Сигнал или поток таймера должен обладать более высоким приоритетом, чем сигнал (поток) обработчика CAN кадров и ошибок CAN контроллера. CANopen таймер может быть не самоблокирующим, то есть возможны повторно-входимые вызовы обработчика таймера. Это дает возможность контролировать наложение тиков таймера при высокой загрузке системы. Обработчик ***handler** является сигнало-безопасным и может быть назначен непосредственно на аппаратные прерывания, в том числе не самоблокирующие.

Если таймер исполняется как отдельный поток операционной системы, метод работы диспетчера ОС может не гарантировать непрерывного выполнения этого потока. В таком случае рекомендуется формировать код обработчика таймера (функция canopen_timer() модуля can_backinit.c) как единую критическую секцию.

Параметры:

- **handler** – функция обработчика таймера, имеет прототип: void canopen_timer(void).

void can_cancel_system_timer(void);

Отмена CANopen таймера. Прекращает либо завершает работу таймера.

void init_critical(void);
void enter_critical(void);
void leave_critical(void);

Функции обслуживания критических секций: инициализация, вход и выход. Служат для обеспечения атомарности семафорных операций и непрерывности сегментов кода при использовании библиотеки в многопоточной среде, когда CANopen таймер и обработчик CAN кадров запускаются как отдельные потоки (нити). Такая ситуация возникает, например, при использовании операционной системы Windows. Функции внедряются в код библиотеки с помощью макросов CAN_CRITICAL_INIT, CAN_CRITICAL_BEGIN и CAN_CRITICAL_END, определенных в модуле can_macros.h. Для однопоточных приложений (микроконтроллеры, операционные системы с поддержкой сигналов) код библиотеки обеспечивает возможность работы с не атомарными семафорами. Таким образом, эти макросы могут оставаться пустыми.

void enable_can_transmitter(void);
void disable_can_transmitter(void);

Функции разрешения работы и блокировки передающего CAN трансивера. Служат для исключения выдачи CAN контроллером в сеть ложных сигналов при включении питания устройства. Работа трансивера разрешается при инициализации CAN подсистемы библиотеки (модуль can_backinit.c).

Модуль светодиодной индикации

Индикация состояния NMT slave устройства осуществляется в соответствии с «проектными рекомендациями по использованию светодиодов» (CiA 303 часть 3 v. 1.4). Для этого используются либо два светодиода: красный (индикация ошибок) и зеленый (индикация работы), либо совмещенный красно/зеленый светодиод. Тип светодиода настраивается параметром сборки приложения CAN_LED_INDICATOR. При использовании совмещенного красно/зеленого светодиода в случае конфликтов индикации преимущество имеет красный светодиод. Для корректной работы светодиодов во всех режимах период CANopen таймера не должен превышать 50 миллисекунд (частота не менее 20 Гц).

Зеленый светодиод (работа)

Индикация	Состояние устройства
Мерцает с частотой 10 Гц в противофазе с красным светодиодом.	Осуществляется автонастройка скорости CAN сети или активирован сервис установки уровня LSS.
Мигает с частотой 2.5 Гц.	Устройство в ПРЕД–операционном состоянии.
Вспышки длительностью 200 мс с паузой 1 с.	Устройство остановлено.
Две вспышки длительностью 200 мс с интервалом 200 мс и паузой 1 с.	Зарезервировано.
Три вспышки длительностью 200 мс с интервалом 200 мс и паузой 1 с.	Производится загрузка в устройство программного обеспечения.
Четыре вспышки длительностью 200 мс с интервалом 200 мс и паузой 1 с.	Зарезервировано.
Светится непрерывно.	Устройство в операционном состоянии.

Красный светодиод (ошибка)

Индикация	Состояние устройства
Погашен.	Нет ошибки. Красный светодиод гасится при получении NMT slave устройством любой адресованной ему NMT команды из CAN сети.
Мерцает с частотой 10 Гц в противофазе с зеленым светодиодом.	Осуществляется автонастройка скорости CAN сети или активирован сервис установки уровня LSS.
Мигает с частотой 2.5 Гц.	Общая конфигурационная ошибка.
Вспышки длительностью 200 мс с паузой 1 с.	Счетчик(и) ошибок CAN контроллера достиг(ли) уровня предостережения (слишком много искаженных кадров в сети).
Две вспышки длительностью 200 мс с интервалом 200 мс и паузой 1 с.	Истекло время жизни для протокола охраны узла. Произошло событие сердцебиения (heartbeat event) для потребителя.
Три вспышки длительностью 200 мс с интервалом 200 мс и паузой 1 с.	Не получен объект синхронизации SYNC за установленный интервал времени (объект 1006_h).

Четыре вспышки длительностью 200 мс с интервалом 200 мс и паузой 1 с.	Не получено RPDO до истечения его таймера события.
Светится непрерывно.	Устройство отключено от шины (в состоянии bus–off).

Оба светодиода гасятся, если NMT slave устройство получает из CAN сети несуществующую NMT команду. При этом NMT состояние устройства не изменяется.

Функции физического управления светодиодами

Представляют собой функции–заглушки, в тело которых должно быть встроено обращение к регистрам управления светодиодами. Начиная с версии 2.1 библиотеки эти функции вынесены в системно–зависимый модуль.

void green_led_on(void);
Физическое включение зеленого светодиода.
void green_led_off(void);
Физическое отключение зеленого светодиода.
void red_led_on(void);
Физическое включение красного светодиода.
void red_led_off(void);
Физическое отключение красного светодиода.

API функций светодиодной индикации

void set_led_green_on(void);
void set_led_green_off(void);
Включение и отключение зеленого светодиода в непрерывный режим.

void set_led_red_on(void);
void set_led_red_off(void);
Включение и отключение красного светодиода в непрерывный режим.

void set_leds_flickering(void);
Включение светодиодов в режим мерцания с частотой 10 Гц. Мерцание красного и зеленого светодиодов осуществляется в противофазе.

void set_led_green_blinking(void);
Включение зеленого светодиода в режим мигания с частотой 2.5 Гц.

void set_led_red_blinking(void);
Включение красного светодиода в режим мигания с частотой 2.5 Гц.

void set_led_green_single_flash(void);
void set_led_green_double_flash(void);
void set_led_green_triple_flash(void);
void set_led_green_quadruple_flash(void);
Включение зеленого светодиода в режим соответственно одной, двух, трех и четырех вспышек длительностью 200 мс с интервалом 200 мс и паузой 1 с.

void set_led_red_single_flash(void);
void set_led_red_double_flash(void);
void set_led_red_triple_flash(void);

void set_led_red_quadruple_flash(void);

Включение красного светодиода в режим соответственно одной, двух, трех и четырех вспышек длительностью 200 мс с интервалом 200 мс и паузой 1 с.

Примеры использования библиотеки

Примеры работы с библиотекой для профиля тестового устройства приведены в модулях:

- \master__can_test_application.c – операции клиента и отображение словаря тестового устройства.
- \slave__obdms_slave_test.h – объектный словарь slave для профиля тестового устройства.
- \master__obdms_master_test.h – объектный словарь мастер–отображения для профиля тестового устройства.

Все функции этих модулей снабжены подробным комментарием.

Номер CAN узла и индекс битовой скорости

Номер CAN узла

Номер узла	Использование
1..127	Номера узлов штатных CANopen устройств.
255	Не сконфигурированное CANopen устройство.

Стандартный набор битовых скоростей CiA

Селектор таблицы стандартных скоростей CAN шины имеет значение 0 (ноль).
Индексы таблицы стандартных скоростей CiA могут принимать следующие значения:

Значение индекса	Скорость CAN сети
0	1 Мбит/с
1	800 Кбит/с
2	500 Кбит/с
3	250 Кбит/с
4	125 Кбит/с
5	зарезервирован
6	50 Кбит/с
7	20 Кбит/с
8	10 Кбит/с
9	автоопределение скорости

Коды ошибок CANopen

Коды ошибок при SDO обмене (SDO аборт код)

Аборт код	Описание
0503 0000$_h$	Не изменился мерцающий (toggle) бит.
0504 0000$_h$	Таймаут SDO протокола.
0504 0001$_h$	Неверная либо не известная команда протокола.
0504 0002$_h$	Неверный размер блока данных (только для блочного протокола).
0504 0003$_h$	Неверный номер кадра (только для блочного протокола).
0504 0004$_h$	Ошибка CRC (только для блочного протокола).
0504 0005$_h$	Не хватает памяти.
0601 0000$_h$	Запрашиваемый доступ к объекту не поддерживается.
0601 0001$_h$	Попытка чтения только записываемого (WO) объекта.
0601 0002$_h$	Попытка записи только читаемого (RO) объекта.
0602 0000$_h$	Нет такого объекта в объектном словаре.
0604 0041$_h$	Объект не может быть отображен в PDO или SRDO.
0604 0042$_h$	Полная длина отображаемых объектов превышает максимальный размер PDO или SRDO (64 бита).
0604 0043$_h$	Общая несовместимость параметров.
0604 0047$_h$	Общая внутренняя несовместимость в устройстве.
0606 0000$_h$	Отказ в доступе из-за аппаратной ошибки.
0607 0010$_h$	Неподходящий тип данных или длина параметра.
0607 0012$_h$	Неподходящий тип данных, превышена длина параметра.
0607 0013$_h$	Неподходящий тип данных, мала длина параметра.
0609 0011$_h$	Нет такого субиндекса.
0609 0030$_h$	Неверное значение параметра (только для записи данных).
0609 0031$_h$	Значение параметра слишком велико (только для записи данных).
0609 0032$_h$	Значение параметра слишком мало (только для записи данных).
0609 0036$_h$	Максимальное значение меньше минимального.
060A 0023$_h$	Ресурс не доступен: SDO соединение.
0800 0000$_h$	Общая ошибка.
0800 0020$_h$	Данные не могут быть переданы приложению.
0800 0021$_h$	Данные не могут быть переданы приложению из-за особенностей локального управления.
0800 0022$_h$	Данные не могут быть переданы приложению вследствие текущего состояния устройства.
0800 0023$_h$	Не удалось динамически сгенерировать объектный словарь или нет

	объектного словаря.
0800 0024$_h$	Нет данных.

Классы ошибок объекта EMCY

Код ошибки	Назначение
00xx$_h$	Сброс либо отсутствие ошибки.
10xx$_h$	Общая ошибка.
20xx$_h$	Ток.
21xx$_h$	Ток на входе в устройство.
22xx$_h$	Ток внутри устройства.
23xx$_h$	Выходной ток устройства.
30xx$_h$	Напряжение.
31xx$_h$	Напряжение питания.
32xx$_h$	Напряжение внутри устройства.
33xx$_h$	Выходное напряжение.
40xx$_h$	Температура.
41xx$_h$	Температура окружающей среды.
42xx$_h$	Температура устройства.
50xx$_h$	«Железо» устройства.
60xx$_h$	Программное обеспечение устройства.
61xx$_h$	Встроенное программное обеспечение.
62xx$_h$	Программное обеспечение пользователя.
63xx$_h$	Данные.
70xx$_h$	Дополнительные модули.
80xx$_h$	Мониторинг.
81xx$_h$	Коммуникации.
82xx$_h$	Ошибка протокола.
90xx$_h$	Внешняя ошибка.
F0xx$_h$	Дополнительные функции.
FFxx$_h$	Определяется конкретным типом CANopen устройства.

Коды ошибок объекта EMCY

Код ошибки	Назначение
0000$_h$	Сброс либо отсутствие ошибки.
1000$_h$	Общая ошибка.
2000$_h$	Ток – общая ошибка.

2100_h	Ток на входе в устройство – общая ошибка.
2200_h	Ток внутри устройства – общая ошибка.
2300_h	Выходной ток устройства – общая ошибка.
3000_h	Напряжение – общая ошибка.
3100_h	Напряжение питания – общая ошибка.
3200_h	Напряжение внутри устройства – общая ошибка.
3300_h	Выходное напряжение – общая ошибка.
4000_h	Температура – общая ошибка.
4100_h	Температура окружающей среды – общая ошибка.
4200_h	Температура устройства – общая ошибка.
5000_h	«Железо» устройства – общая ошибка.
6000_h	Программное обеспечение устройства – общая ошибка.
6100_h	Встроенное программное обеспечение – общая ошибка.
6180_h	Переполнение выходного CANopen кэша.
6190_h	Ошибка инициализации CANopen таймера.
6191_h	Наложение тиков CANopen таймера.
$61A0_h$	Ошибка данных в энергонезависимой памяти (неверный CRC).
$61A1_h$	Ошибка при работе с энергонезависимой памятью.
6200_h	Программное обеспечение пользователя – общая ошибка.
6300_h	Данные – общая ошибка.
7000_h	Дополнительные модули – общая ошибка.
8000_h	Мониторинг – общая ошибка.
8100_h	Коммуникации – общая ошибка.
8110_h	Переполнение CAN (потеря объекта).
8120_h	CAN в пассивном к ошибке состоянии.
8130_h	Ошибка протокола сердцебиения либо охраны узла.
8140_h	Выход из состояния отключения от шины (bus–off).
8150_h	Коллизия передаваемых идентификаторов (CAN–ID).
8180_h	Событие CAN контроллера «hardware overrun».
8181_h	Событие CAN контроллера «software overrun».
8182_h	Событие CAN контроллера «error warning limit».
8183_h	Событие CAN контроллера «write timeout».
8190_h	Прекращена работа по безопасному протоколу EN50325-5.
8200_h	Ошибка протокола – общая ошибка.
8210_h	PDO не может быть обработан из–за ошибки длины данных.
8211_h	SRDO не может быть обработан из-за ошибки длины данных.

8220_h	Превышен превышает максимальный размер PDO.
8230_h	Не обработан мультиплексированный PDO с режимом адреса назначения (DAM): соответствующий объект не доступен.
8240_h	Неподходящая длина данных SYNC кадра.
8250_h	Таймаут RPDO.
9000_h	Внешняя ошибка – общая ошибка.
$F000_h$	Дополнительные функции – общая ошибка.
$FF00_h$	Определяется конкретным типом CANopen устройства – общая ошибка.

Цветом выделены дополнительные и не стандартные коды ошибок.
Ошибки с кодами 6180_h, 6190_h, $61A0_h$ и $61A1_h$ заносятся в список ошибок (объект 1003_h) но не передаются в качестве срочного сообщения, поскольку объект EMCY отсутствует в системе (этап инициализации) либо не может быть передан в CAN сеть.

Предопределенное распределение идентификаторов

Широковещательные объекты

Идентификаторы широковещательных объектов не зависят от номера CAN узла.

CAN–ID	Назначение	Индекс объекта
0	NMT объекты.	——
1	GFC команда (EN50325-5).	1300_h
128 (80_h)	Объект синхронизации SYNC.	1005_h
256 (100_h)	Объект временной метки TIME.	1012_h

Объекты класса равный–к–равному (peer–to–peer)

Идентификаторы объектов равный–к–равному зависят от номера CAN узла.

CAN–IDs	Назначение	Индекс объекта
129 (81_h) – 255 (FF_h)	Объекты срочного сообщения EMCY для узлов сети 1 – 127.	1014_h
257 (101_h) – 384 (180_h)	Объекты данных безопасного протокола (SRDO, EN50325-5)	1301_h
385 (181_h) – 511 ($1FF_h$)	Первые передаваемые PDO (TPDO1) для узлов сети 1 – 127.	1800_h
513 (201_h) – 639 ($27F_h$)	Первые принимаемые PDO (RPDO1) для узлов сети 1 – 127.	1400_h
641 (281_h) – 767 ($2FF_h$)	Вторые передаваемые PDO (TPDO2) для узлов сети 1 – 127.	1801_h
769 (301_h) – 895 ($37F_h$)	Вторые принимаемые PDO (RPDO2) для узлов сети 1 – 127.	1401_h
897 (381_h) – 1023 ($3FF_h$)	Третьи передаваемые PDO (TPDO3) для узлов сети 1 – 127.	1802_h
1025 (401_h) – 1151 ($47F_h$)	Третьи принимаемые PDO (RPDO3) для узлов сети 1 – 127.	1402_h
1153 (481_h) – 1279 ($4FF_h$)	Четвертые передаваемые PDO (TPDO4) для узлов сети 1 – 127.	1803_h
1281 (501_h) – 1407 ($57F_h$)	Четвертые принимаемые PDO (RPDO4) для узлов сети 1 – 127.	1403_h
1409 (581_h) – 1535 ($5FF_h$)	SDO, передаваемые от сервера клиенту для узлов сети 1 – 127.	1200_h
1537 (601_h) – 1663 ($67F_h$)	SDO, передаваемые от клиента серверу для узлов сети 1 – 127.	1200_h
1793 (701_h) – 1919 ($77F_h$)	Протоколы контроля ошибок (сердцебиения и охраны узла) для узлов сети 1 – 127.	1016_h, 1017_h

Прочие объекты

CAN–ID	Назначение
2020 ($7E4_h$)	Ответ от LSS slave (сервис установки уровня).
2021 ($7E5_h$)	Запрос от LSS master (сервис установки уровня).

Идентификаторы ограниченного использования

Идентификаторы ограниченного использования не должны применяться в любых конфигурируемых коммуникационных объектах, будь то SYNC, TIME, EMCY, PDO или дополнительные SDO.

CAN–IDs	Назначение
0	NMT объекты.
1	GFC команда (EN50325-5).
2 (002_h) – 127 ($07F_h$)	Зарезервированы.
257 (101_h) – 384 (180_h)	Объекты данных протокола EN50325-5 (SRDO).
1409 (581_h) – 1535 ($5FF_h$)	SDO по умолчанию, передаваемые от сервера клиенту.
1537 (601_h) – 1663 ($67F_h$)	SDO по умолчанию, передаваемые от клиента серверу.
1760 ($6E0_h$) – 1791 ($6FF_h$)	Зарезервированы.
1793 (701_h) – 1919 ($77F_h$)	Протоколы контроля ошибок.
1920 (780_h) – 2047 ($7FF_h$)	Зарезервированы.

Тест Соответствия – CANopen conformance test

Тест Соответствия (CANopen conformance test plan, CiA 310) предназначен для проверки устройств, использующих протокол CANopen. Тестированию на соответствие стандарту подвергается поведение устройства в качестве узла CAN сети. Внутренняя логика работы устройства (прикладной профиль) проверке с помощью Теста не подлежит. Любое устройство, поддерживающее протокол CANopen, должно пройти проверку (сертификацию) с использованием Теста Соответствия.

Программное обеспечение Теста Соответствия CANopen распространяется организацией CAN in Automation. Для доступа к сети Тест использует стандартизованный набор функций, называемый COTI (CANopen Test Interface). Для того чтобы Тест работал с тем или иным CAN интерфейсом, производитель должен предоставить библиотеку COTI для своих CAN адаптеров.

Тест Соответствия реализует следующие операции:

- Проверку электронной спецификации устройства (Electronic Data Sheet – EDS) на соответствие стандарту CiA 306.
- Тестирование сетевого протокола на соответствие стандарту CiA 301. Используются 11-битовые CAN идентификаторы и предопределенное распределение этих идентификаторов (Pre–Defined Connection Set).
- Проверку соответствия объектного словаря устройства его электронной спецификации.

Для ряда прикладных CANopen профилей возможна полная проверка EDS файла устройства по соответствующей базе данных профиля. Такие базы разработаны фирмой Vector Informatik GmbH и доступны на сайте CiA.

С 2013 года доступна третья главная версия Теста Соответствия, которая поддерживает обновленные стандарты CiA. В ряде случаев новая версия Теста осуществляет более строгие проверки протоколов и объектного словаря CANopen устройства. CANopen библиотека Марафон версий 2.3 и выше адаптирована для прохождения Теста Соответствия третьей версии.