Утверждаю
Директор
ООО «Сигма-Софт
Автоматизация»
_____М.И. Мальцев
“_____”_____2023

Программный комплекс «С-Платформа» (S-Platform)

Руководство программиста

RU.82469608.0001-01 33

Том 1

Подсистема коммуникаций

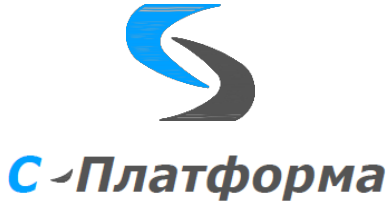
Версия 1.6

Руководитель разработки
Начальник департамента

_____И.О. Урухин
“_____”_____2023 г.

Ответственный исполнитель
Ведущий инженер-программист
_____ В.Э. Форш
“_____”_____2023 г.

ООО «Сигма Софт»
2023 г.



Утвержден

RU.82469608.0001-01 33

Программный комплекс «С-Платформа» (S-Platform)

Руководство программиста (том 1)

RU.82469608.0001-01 33

Том 1

Подсистема коммуникаций

Версия 1.6

Листов 26

ООО «Сигма Софт»
2023 г.

АННОТАЦИЯ

Настоящий документ содержит руководство программиста на подсистему коммуникаций программного комплекса «С-Платформа» - серверу ввода-вывода Rdx (далее по тексту – Сервер Rdx).

Ядром подсистемы является сервер ввода-вывода из состава программного комплекса (ПК) КОТМИ-14. Сервер ввода-вывода служит контейнером и диспетчером для программ-протоколов информационного обмена (драйверов). Программы-протоколы обеспечивают коммуникации с различными устройствами, контроллерами, локальными САУ, смежными системами управления различного уровня.

В данном руководстве описан интерфейс взаимодействия сервера Rdx с протокольными модулями. А также правила создания шаблонов с конфигурационными параметрами протоколов.

СОДЕРЖАНИЕ

Лист

1. СТРУКТУРА СЕРВЕРА RDX

Программа RdxServer.exe (RdxServer – в Linux) состоит из следующих основных частей:

- ядро системы;
- разборщик (парсер) конфигурационного файла;
- набор модулей, реализующих канальные уровни разных протоколов;
- набор модулей, реализующих прикладные уровни разных протоколов.

Модули прикладных протоколов - отдельные файлы (dll, libRdxX), взаимодействующие с сервером ввода-вывода через специальный интерфейс расширения.

Функционирование сервера ввода-вывода подразделяется на три этапа:

Старт и конфигурирование системы (start time). На этом этапе работает один, основной поток процесса. Читается и разбирается конфигурационный xml-файл, заданный в параметрах командной строки. В соответствии с заданной конфигурацией формируется внутренняя структура направлений, каналов, параметров обмена и связей между ними. Настраивается требуемая схема резервирования для каналов, работающих в составе одного направления. После того как структура сформирована, последовательно запускаются на выполнение все потоки каналов ввода-вывода.

Основной цикл обработки (run time). Запущены и работают каналы ввода-вывода. В зависимости от состояния каналов (подключен-отключен), осуществляется прием данных, ретрансляция и передача, в соответствии со схемой стартовой конфигурации. Каждый канал обслуживается отдельным, событийно-управляемым потоком процесса. Основное состояние потока – это режим ожидания события (“спящий режим”). Переход потока из “спящего” состояния в активное происходит по следующим событиям: событие таймера протокола, принятие данных из канала, появление данных для маршрутизации, завершение работы.

Останов системы (stop time). После того, как основной поток сервиса получил событие «*завершение работы*» он выполняет следующие действия: устанавливает события «*завершение работы*» для потоков всех каналов, освобождает все, ранее созданный программные объекты, завершает работу процесса.

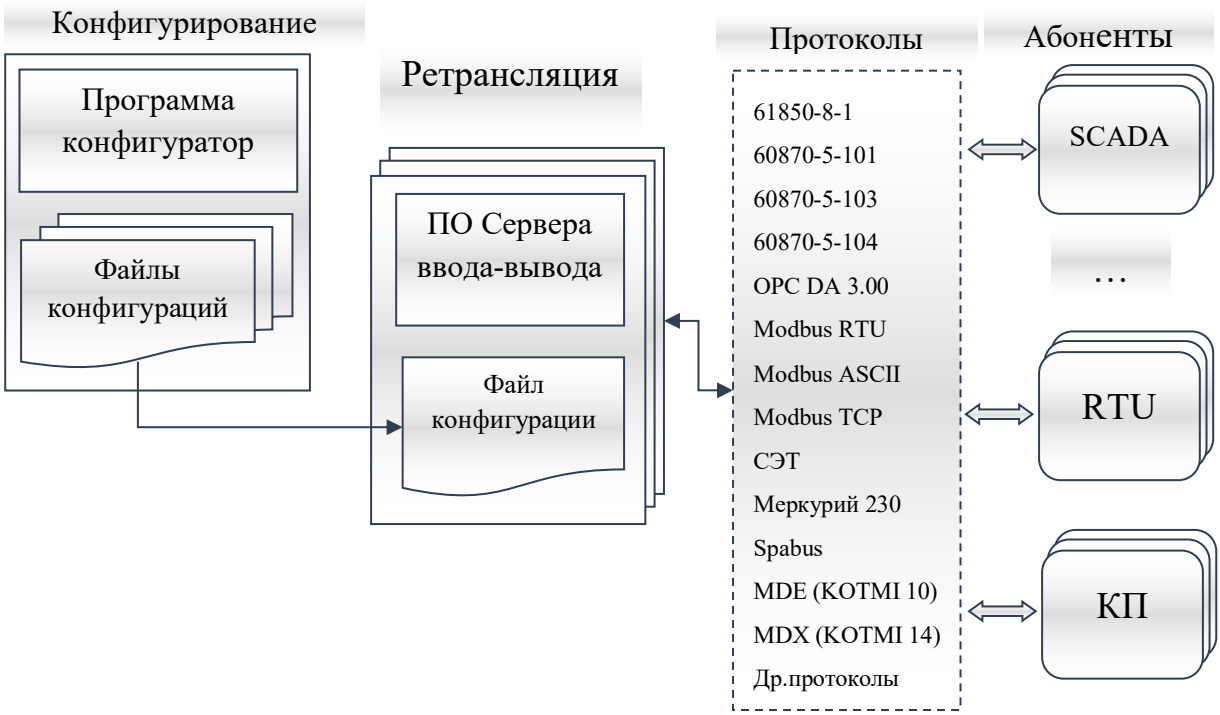


Рисунок 1 - Основные компоненты и взаимосвязи

2. ШАБЛОН ПРОТОКОЛА

Шаблон протокола содержит перечень всех конфигурационных параметров. Параметры делятся на три уровня: уровень линии, уровень устройства и уровень сигналов. Каждому уровню соответствует свой тег в шаблоне. Тег <protocol> соответствует уровню линии. Тег <remoteunit> соответствует уровню устройства. Уровню сигналов соответствуют теги <source>, <pass>, <controlsource>, <controlpass>, соответственно для принимаемых сигналов, передаваемых сигналов, принимаемых команд и передаваемых команд.

```
<protocol>

 <remoteunit>

  <source> </source>
  <pass> </pass>
  <controlsource> </controlsource>
  <controlpass> </controlpass>

 </remoteunit>

</protocol>
```

В теге <protocol> необходимо перечислить все конфигурационные параметры уровня линии. И указать значение по умолчанию для каждого параметра.

Пример:

```
<protocol
  name="Sample" caption="Пример" module="RdxSample" driver="Tcp"
  redundancy="sample" priority="0" mode="cli" IPaddress="127.0.0.1" port="2000"
  portLocal="2001" portNumber="1" baudRate="19200" dataBits="8" parity="0"
  stopBits="0" portCfg="0" timeoutCfg="0" tuwait="10000" timeShiftS="0"
  timeShiftW="0" startDelay="0" overlapped="1" t0="3000" t1="0" t2="100" t14="50">
```

Каждый параметр имеет отдельный тег <property>. В этом теге необходимо указать: имя, название, тип, режим редактирования и видимость параметра. А также указать является ли этот параметр ключевым для формирования адреса.

Пример:

```
<property name="port" caption="TCP-порт" type="int" editing="true" visible="true"
key="false"/>
```

В теге <remoteunit> необходимо перечислить все конфигурационные параметры уровня устройства. И указать значение по умолчанию для каждого параметра. Каждый параметр имеет отдельный тег <property>. В этом теге необходимо указать: имя, название, тип, режим редактирования и видимость параметра. А также указать является ли этот параметр ключевым для формирования адреса.

В тегах <source>, <pass>, <controlsource>, <controlpass> необходимо перечислить все конфигурационные параметры уровня сигналов. И указать значение по умолчанию для каждого параметра. Каждый параметр имеет отдельный тег <property>. В этом теге необходимо указать: имя, название, тип, режим редактирования и видимость параметра. А также указать является ли этот параметр ключевым для формирования адреса.

Шаблон протокола должен находиться в подкаталоге protocols, рабочего каталога сервера Rdx. Программа Конфигуратор, при старте, считывает из этого каталога все доступные шаблоны. Если в каталоге protocols нет шаблона соответствующего протокола, то программа Конфигуратор не сможет создать линию с этим протоколом. В процессе работы сервера Rdx, файл с шаблоном протокола не используется.

3. ПРОТОКОЛЬНЫЙ МОДУЛЬ

3.1. Общие положения.

Протокольный модуль (далее по тексту – ПМ) представляет собой динамически подключаемую библиотеку. Файл с расширением dll для ОС Windows или файл с расширением so для ОС Линукс. Имя этого ПМ должно быть указано в атрибуте module файла с шаблоном этого протокола.

<protocol module="RdxSample">

Взаимодействие ПМ с сервером Rdx может осуществляться двумя способами. Либо посредством производного класса, либо при помощи интерфейсных функций. Базовый класс `CRdxProtocol` объявлен в файле mod.h. Перечень интерфейсных функций указан в файле core.h.

3.2. Подключение протокольного модуля к серверу Rdx.

Каждый ПМ должен переопределить функции RdxInit и RdxRun. Сервер Rdx вызывает функцию RdxInit для того чтобы создать объект ПМ. И функцию RdxRun, для того чтобы запустить в работу созданный объект.

Пример:

```
#ifdef __cplusplus
extern "C" {
#endif

void * _std_ RdxInit(rdxLine * line, rdxSrvInterface * srvInterface)
{
  CSample * obj = new CSample(line, srvInterface);
  if ( !obj->Init() )
  {
    delete obj;
    return 0;
  }
  return obj;
}

void _std_ RdxRun(void * protocol)
{
  ((CSample *) protocol)->Run();
  delete (CSample *) protocol;
}

#include "_rev.h"

_exp_ void _std_ RdxVersion(
  long *Major,
  long *Minor,
  long *Release,
  long *Build,
  const char **Author)
{
  if (Major) { *Major = 2; }
  if (Minor) { *Minor = 1; }
  if (Release) { *Release = 0; }
  if (Build) { *Build = REV_BUILD; }
  if (Author) { *Author = "Pushkin A. S."; }
}

#ifdef __cplusplus
}
#endif
```

3.3. Декларация производного класса.

В ПМ необходимо определить класс, производный от базового класса CRdxProtocol. Базовый класс CRdxProtocol объявлен в файле mod.h. В производном классе надо переопределить следующие функции:

- Init, вызывается из функции RdxInit
- Run, вызывается из функции RdxRun
- ProcessingMessage, обработка сообщений к линии
- ProcessingRouter, обработка данных поступающих по ретрансляции
- ProcessingOutput, передача пакетов в канал
- ProcessingInput, прием пакетов из канала
- ProcessingTimeout, обработка таймаутов протокола
- Connect, открытие соединения
- Disconnect, закрытие соединения

Пример:

```
class CSample: public CRdxProtocol
{
// переменные и функции класса
public:
  CSample (rdxLine *, rdxSrvInterface *);
  ~CSample ();

  bool Init();
  void Run();

  void ProcessingRouter();
  void ProcessingOutput();
  void ProcessingInput();
  void ProcessingTimeout(rdxInt32);

  void Connect();
  void Disconnect(bool);
};
```

4. ОПРЕДЕЛЕНИЕ ПРОИЗВОДНОГО КЛАССА

4.1. Функция Init

Вызывается при старте сервераRdx. Выполняет начальную инициализацию линии и проверку конфигурационных параметров. Если функция Init возвращает false, то линия не стартует.

Пример:

```
bool CSample::Init()
{
  bool rc = false;
  if ( CRdxProtocol::Init() )
  {
// сделать что-то
    rc = true;
  }

  return rc;
}
```

4.2. Функция Run

Основной цикл работы линии. Из этой функции вызываются все остальные обработчики: ProcessingMessage, ProcessingRouter, ProcessingOutput, ProcessingInput, Disconnect, Connect, ProcessingTimeout.

Пример:

```
void CSample::Run()
{
  // Создание драйвера
  m_drv = CreateDriver(true);
  // Основной цикл обработки
  m_sapi->RdxLineState(m_line, IsActive());
  if ( !IsStopped() )
  {
    Connect();
  }
  while (!(m_line->flags & RDX_LINE_FINISH))
  {
    m_actions = false;
    // Обработка событий
    if ( m_line->msgCount )
    {
      ProcessingMessage();
    }
    // Обработка таймеров
    rdxClock timeout = m_sapi->RdxTimerTest(m_line);
    if ( m_sapi->RdxTimerExpire(m_line) )
    {
      ProcessingTimeout(m_sapi->RdxTimerExpGet(m_line));
      m_actions = true;
    }
    // Тестирование состояния драйвера ввода-вывода
    m_sapi->RdxDriverState(m_drv);
```

```
    if ( !(m_drv->flags & RDX_DRIVER_ACTIVE) )
    {
      if ( m_line_active == true )
      {
        Disconnect(false);
      }
    }
    // Передача в канал
    if ( (m_drv->flags & RDX_DRIVER_WRITE) && IsActive() )
    {
      ProcessingOutput();
      m_actions = true;
    }
    // Прием из канала
    if ( IsConnected() )
    {
      ProcessingInput();
    }
    // Обработка ретрансляции из других протоколов
    if ( m_line->valCount && (m_drv->flags & RDX_DRIVER_WRITE) && IsActive() )
    {
      ProcessingRouter();
      m_actions = true;
    }
    // Управление подключениями
    if ( IsStopped() )
    {
      if ( IsActive() )
      {
        Disconnect(m_line->flags & (RDX_LINE_STOP | RDX_LINE_FINISH));
      }
    }
    else if ( !IsActive() && !(m_sapi->RdxTimerActive(m_line) & (1 << TIMER_OPEN)) )
    {
      m_sapi->RdxTimerSet(m_line, TIMER_OPEN, m_open_timeout);
      m_actions = true;
    }
    // Оповещение о состоянии
    if ( CalcClockDlt(m_sapi->RdxClockNow(), m_line->stateTime) > STATUS_TEST_TIMEOUT )
    {
      if ( IsActive() == true )
      {
        m_sapi->RdxLineState(m_line, true);
      }
      else
      {
        m_sapi->RdxLineState(m_line, false);
      }
    }
    if ( m_actions )
    {
      continue;
    }
    // Ожидание
    if ( timeout > STATUS_TEST_TIMEOUT )
    {
      timeout = STATUS_TEST_TIMEOUT;
    }
    m_wait_result = m_sapi->RdxWait(m_line, timeout);
  }
  // Завершение работы
  if ( IsActive() )
  {
    Disconnect(true);
  }
}
```

4.3. Функция ProcessingMessage

Обрабатывает сообщения, поступающие от других линий.

4.4. Функция ProcessingRouter

Обрабатывает данные, поступающие по ретрансляции от других линий.

Пример:

```
void CSample::ProcessingRouter()
{
  rdxTag * tag = 0;
  rdxVariant * val = 0;
  while (1)
  {
    val = m_sapi->RdxReadVal(m_line, &tag);
    if ( val )
    {
    }
    else
    {
      break;
    }
  }

  return;
}
```

4.5. Функция ProcessingOutput

Передает массив байт в драйвер.

Пример:

```
void CSample::ProcessingOutput()
{
  rdxInt32 len = m_sapi->RdxDriverSend(m_drv,
                                       m_write_buff.m_data.data(),
                                       m_write_buff.m_data_len, 10);
  return;
}
CRdxIOBuffer CRdxProtocol::m_write_buff // буфер для взаимодействия с драйвером
```

4.6. Функция ProcessingInput

Принимает массив байт из драйвера.

Пример:

```
void CSample::ProcessingInput()
{
  rdxInt32 len = m_sapi->RdxDriverSend(m_drv,
                                       m_read_buff.m_data.data(),
                                       m_read_buff.m_data_len, 10);
  return;
}
CRdxIOBuffer CRdxProtocol:: m_read_buff // буфер для взаимодействия с драйвером
```

4.7. Функция Disconnect

Выполняет все необходимые операции при закрытии соединения.

Пример:

```
void CSample::Disconnect(bool isStop)
{
  m_sapi->RdxTimerClear(m_line);
  m_sapi->RdxLineFlags(m_line, (rdxUInt32) RDX_LINE_TEST, false);
  CRdxProtocol::Disconnect(isStop);

  return;
}
```

4.8. Функция Connect

Устанавливает соединение.

Пример:

```
void CSample::Connect()
{
  CRdxProtocol::Connect();
  m_sapi->RdxDriverOpen(m_drv);
  while (true)
  {
    m_sapi->RdxDriverState(m_drv);
    if (m_line->flags & RDX_LINE_FINISH)
    {
      break;
    }
    if ( !(m_drv->flags & RDX_DRIVER_CONNECTING) )
    {
      break;
    }
    if ( CalcClockDlt(m_sapi->RdxClockNow(), m_line->stateTime) > STATUS_TEST_TIMEOUT )
    {
       m_sapi->RdxLineState(m_line, IsActive());
    }
    m_sapi->RdxWait(m_line, 10);
  }
  if ( IsConnected() )
  {
// сделать что-то
  }

  return;
}
```

4.9. Функция ProcessingTimeout

Обрабатывает таймауты протокола.

Пример:

```
void CShttp::ProcessingTimeout(rdxInt32 timer)
{
  switch (timer)
  {
    case 1:
    {
    }
    break;

    default:
    {
      CRdxProtocol::ProcessingTimeout(timer);
    }
    break;
  }
}
```

5. ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ БАЗОВОГО КЛАССА

Также, в базовом классе определены следующие вспомогательные функции;

- IsMain, признак основной линии
- IsDataTransmission, признак режима обмена данными
- IsActive, признак работоспособности линии
- IsConnected, признак наличия открытого канала
- IsFinished, признак завершения потока обработки линии
- IsStopped, признак принудительного останова линии
- IsPrint, признак и печать сообщения указанного уровня логирования

6. ИНТЕРФЕЙСНЫЕ ФУНКЦИИ

Протокольный модуль может быть разработан не на основе базового класса `CRdxProtocol`. ПМ может взаимодействовать с ядром сервера Rdx посредством интерфейсных функций. Интерфейсные функции объявлены в файле core.h.

6.1. Функции записи ретранслируемых тегов

```
RdxWrite(rdxLine *line, rdxUInt32 tagType, rdxParamValue *key, rdxVariant *data) // Ретрансляция значения (поиск тэга по ключу)
RdxWriteToTag(rdxTag *tag, rdxVariant *data) // Ретрансляция значения
RdxWriteMsg(rdxLine *line, rdxVariant *data, void *obj, rdxByte cot, rdxChar *note) // Передача сообщения линии
```

6.2. Функции чтения ретранслируемых тегов

```
RdxReadVal(rdxLine *line, rdxTag **tag) // Выборка значений из очереди линии
RdxReadFromTag(rdxTag *tag) // Выборка значений из очереди тэга
RdxReadMsg(rdxLine *line, rdxVariant **data, void **obj, rdxByte *cot, rdxChar *note) // Выборка сообщения из очереди линии
```

6.3. Функции ожидания событий и временных интервалов

```
RdxWait(rdxLine *line, rdxClock timeout) // Ожидание события
RdxSleep(rdxUInt32 timeout) // Задержка потока на указанное время
```

6.4. Функции передачи состояния линии и устройства

```
RdxLineState(rdxLine *line, rdxBool isActive) // Управление состоянием линии (результат-статус линии)
RdxUnitState(rdxUnit *unit, rdxBool isActive) // Управление состоянием устройства (результат-статус линии)
```

6.5. Функции записи записи в лог-файлы

```
RdxPrint(rdxLine *line, rdxUInt32 level, const rdxChar *func, const rdxChar *mess, const rdxByte* frame, rdxUInt32 frameLen) // Вывод в лог-файл линии
RdxPrintConfig(rdxConfig *config, rdxUInt32 level, const rdxChar *func, const rdxChar *mess) // Вывод в лог-файл конфигурации
```

6.6. Функции поиска линии, устройства, тега

```
RdxLineByNum(rdxConfig *config, rdxUInt32 lineNum) // Поиск линии по номеру
RdxUnitByNum(rdxConfig *config, rdxUInt32 unitNum) // Поиск устройства по номеру
RdxTagByNum(rdxConfig *config, rdxUInt32 tagNum) // Поиск тэга по номеру
RdxTagByKey(rdxLine *line, rdxUInt32 tagType, rdxParamValue *key) // Поиск тэга линии по ключу
```

```
RdxTagVal(rdxTag *tag) // Текущее значение тэга
```

6.7. Функции телеуправления

```
RdxCmdData(rdxConfig *config, rdxUInt32 dstNum, rdxUInt32 *srcNum) // Получение данных команды ТУ по номеру тэга приемника
```

6.8. Функции установки флагов

```
RdxLineFlags(rdxLine *line, rdxUInt32 mask, rdxBool include) // Установка или сброс флагов линии
RdxTagFlags(rdxTag *tag, rdxUInt32 mask, rdxBool include) // Установка или сброс флагов тэга
RdxWaitFlags(rdxLine *line, rdxUInt32 mask) // Установка флагов и вывод протокола из состояния ожидания
```

6.9. Функции для работы со структурой rdxVariant

```
RdxVarNew(void) // Получение нового rdxVariant
RdxVarDel(rdxVariant *value) // Освобождение rdxVariant
RdxVarDup(rdxVariant *value) // Увеличение счетчика ссылок rdxVariant
RdxVarCopy(rdxVariant *dst, rdxVariant *src) // Копирование данных
RdxVarBlobSet(rdxVariant *value, const rdxByte *data, rdxUInt32 dataLen) // Запись длинной строки в rdxVariant
RdxVarBlobFree(rdxVariant *value) // Освобождение длинной строки в rdxVariant
```

6.10. Функции для работы с общими данными

```
RdxCommonSet(rdxConfig *config, rdxUInt32 type, rdxUInt32 number, rdxVariant *data) // Задание значений общих данных
RdxCommonGet(rdxConfig *config, rdxUInt32 type, rdxUInt32 number) // Получение значений общих данных
```

6.11. Функции для работы с конфигурационными параметрами

```
RdxParamCount(void *nciParams) // Количество параметров НСИ
RdxParamValue(void *nciParams, rdxUInt32 paramIdx, rdxParamValue *value) // Значение параметра по индексу
RdxParamByName(void *nciParams, const rdxChar *paramName, rdxParamValue *value) // Значение параметра по имени
RdxParamName(void *nciParams, rdxUInt32 paramIdx) // Имя параметра по индексу
RdxParamIdx(void *nciParams, const rdxChar *paramName) // Индекс параметра по его имени (-1 - нет)
RdxParamKeys(rdxTag *tag, rdxParamValue *value) // Получение значений ключевых параметров тэга
```

6.12. Функции для работы с таймерами

```
RdxTimerSet(rdxLine *line, rdxUInt32 number, rdxUInt32 interval) // Установка таймера (0..31)
RdxTimerTest(rdxLine *line) // Проверка. Возвращает min интервал до истечения || RDX_WAIT_INFINITE
RdxTimerExpGet(rdxLine *line) // Возвращает номер очередного сработавшего таймера (-1, если нет)
RdxTimerActive(rdxLine *line) // Возвращает битовую маску активных таймеров
RdxTimerExpire(rdxLine *line) // Возвращает битовую маску сработавших таймеров
RdxTimerClear(rdxLine *line) // Сброс всех таймеров
```

6.13. Функции установки, получения и преобразования структур времени

RdxTimeNow() // Текущее UTC-время (1970-01-01 00:00:00)
RdxTimeMake(rdxSystemTime *time, rdxBool isUtc) // Получение времени из структуры
RdxTimeLocal(rdxTime src, rdxSystemTime *dst) // Получение структуры локального времени
RdxTimeUtc(rdxTime src, rdxSystemTime *dst) // Получение структуры UTC времени
RdxTimeSet(rdxTime time) // Установка системного времени (UTC)
RdxClockNow() // Текущее количество миллисекунд

6.14. Функции для работы с драйвером

RdxDriverNew(rdxLine *line, rdxInt32 type) // Создание нового драйвера для линии
RdxDriverDel(rdxDriver *driver) // Удаление драйвера
RdxDriverOpen(rdxDriver *driver) // Открытие соединения
RdxDriverClose(rdxDriver *driver) // Закрытие соединения
RdxDriverSend(rdxDriver *driver, rdxByte *dataBuff, rdxUInt32 dataLen, rdxClock timeout) // Передача данных в канал
RdxDriverRecv(rdxDriver *driver, rdxByte *dataBuff, rdxUInt32 dataMax, rdxClock timeout) // Прием данных из канала
RdxDriverState(rdxDriver *driver) // Получение флагов текущего состояния драйвера

6.15. Дополнительные функции

RdxSpecialData(rdxConfig *config, rdxUInt32 type, rdxUInt32 number, rdxParamValue *value) // Получение специальных данных конфигурации
RdxReservePost(rdxLine *line, rdxSystemTime time, bool toChannel) // Очистка очереди резервных линий по времени ВУ

7. СТРУКТУРЫ ДАННЫХ ИНТЕРФЕЙСНЫХ ФУНКЦИЙ

Структуры данных, которые используются при вызове интерфейсных функций (core.h).

7.1. Структура системного времени

```
typedef struct {
 rdxUInt32 uYear;       // от РХ
 rdxUInt32 uMonth;      // 1..12
 rdxUInt32 uDay;        // 1..31
 rdxUInt32 uHour;       // 0..23
 rdxUInt32 uMinute;     // 0..59
 rdxUInt32 uSecond;     // 0..59
 rdxUInt32 uMilliseconds;// 0..999
 rdxUInt32 uDayOfWeek;  // 1 = Monday to 7 = Sunday
} rdxSystemTime;
```

7.2. Структура для работы с драйвером

```
struct rdxDriver
{
 rdxUInt32      type;          // Тип драйвера
 rdxLine       *line;          // Линия драйвера
 rdxUInt32      flags;         // Флаги состояния
 rdxInt32       lastError;     // Код ошибки
 rdxUInt64      byteRecv;      // Количество принятых байт
 rdxUInt64      byteSend;      // Количество переданных байт
 rdxTime        timeStart;     // Время открытия соединения
 rdxTime        timeStop;      // Время закрытия соединения
};
```

7.3. Структура конфигурационных параметров

```
struct rdxConfig
{
 rdxUInt32      number;        // Номер конфигурации
 void          *nciParams;             // Адрес строки     параметров
 rdxUInt32      directionCount; // Количество направлений в конфигурации
 rdxDirection  **directionArray; // Массив направлений (сортировка по номеру)
 rdxUInt32      lineCount;     // Количество линий в конфигурации
 rdxLine       **lineArray;    // Массив линий (сортировка по номеру)
 rdxUInt32      unitCount;     // Количество устройств в конфигурации
 rdxUnit       **unitArray;    // Массив устройств (сортировка по номеру)
 rdxUInt32      flags;         // Флаги состояния конфигурации
 rdxUInt32      logLevelFile;        // Уровень логирования
 rdxUInt32      logLevelMon;   // Уровень логирования (монитор)
 rdxDirection   *manualSource;  // Источник ручного ввода
 rdxChar       *fileName;      // Имя файла конфигурации
 rdxLine       *monitorLine;   // Линия для связи с монитором сервера в-в
 rdxClock       statusError;   // Таймаут отсутствия подтверждения для заключения об аварийном состоянии линии
 rdxClock       stateTime;     // Время последнего подтверждения состояния активности потока конфигурации
 rdxTime        startTime;     // Время старта конфигурации (UTC)
 rdxBool        tzStandart;    // Признак зимнего времени
 rdxClock       tzUpdate;      // Время последнего обновления признака зимнего времени
 rdxUInt32      timeReservDepth;// Глубина резервной очереди при резервировании серверов (сек)
};
```

7.4. Структура направления

```
struct rdxDirection
{
 rdxUInt32      number;       // Номер направления
 void          *nciParam      // Адрес строки параметров
 rdxConfig     *config;       // Конфигурация
 rdxUInt32      lineCount;    // Количество линий (резервирование)
 rdxLine      **lineArray;    // Список резервируемых линий данного направления (сортировка по приоритету)
 rdxUInt32      keyCount[4];  // Количество ключевых полей НСИ для тэгов направления
 rdxUInt32     *keyArray[4];  // Массив индексов ключевых полей для тэгов направления
 rdxUInt32      flags;        // Флаги направления
 rdxClock       statusTime;   // Последнее время подтверждения основной линии
 rdxLine       *statusLine;   // Основная линия
};
```

7.5. Структура линии

```
struct rdxLine
{
 rdxUInt32      number;    // Номер
 void          *nciParams; // Адрес строки  параметров
 rdxDirection  *direction; // Направление
 rdxUInt32      unitCount; // Количество устройств
 rdxUnit      **unitArray;  // Список устройств линии (сортировка по номеру)
 rdxUInt32      tagCount[4];  // Количество тэгов в линии (RDX_TAG_SOURCE..RDX_CMD_PASS)
 rdxTag       **tagArray[4];  // Массив тэгов (сортировка по ключу)
 rdxTag        *tagState;     // Тэг состояния протокола (ОТКЛ\ВКЛ)
 rdxTag        *tagStatus;    // Тэг статуса протокола (основной-резервный)
 rdxUInt32      flags;        // Флаги состояния линии
 rdxUInt32      logLevelFile; // Уровень логирования (лог-файл)
 rdxUInt32      logLevelMon; // Уровень логирования (монитор)
 rdxUInt32      priority;     // Приоритет в направлении
 rdxClock       stateTime;   // Время последней синхронизации состояния и статуса
 rdxTime        startTime;   // Время последнего старта линии (UTC)
 rdxInt32       timeShiftS;  // Смещение летнего времени (sec)
 rdxInt32       timeShiftW;  // Смещение зимнего времени (sec)
 rdxUInt32      msgCount;     // Количество элементов в очереди сообщений
 rdxUInt32      msgQueued;    // Количество элементов добавленных в очередь сообщений
 rdxUInt32      valCount;     // Количество элементов в очереди данных
 rdxUInt32      valQueued;    // Количество элементов добавленных в очередь данных
 rdxUInt32      resCount;     // Количество элементов в очереди резервирования
 rdxUInt32      resQueued;    // Количество элементов добавленных в очередь резервирования
 rdxClock       commandWait;  // Таймаут ожидания ТУ (COMMAND_WAIT_DEFAULT, может меняться протоколом)
 rdxTime        valLastTime;  // Время последней ретрансляции значений ???
 rdxUInt64      numberLostTS;  // Количество потерянных ТС
 rdxUInt64      numberLostTI;  // Количество потерянных значений других типов данных
 rdxUInt32      timeExceeding; // Контроль превышения времени сигнала (сек)
 void          *temp;         // Поле может использоваться протоколом
};
```

7.6. Структура устройства

```
struct rdxUnit
{
 rdxUInt32      number;     // Номер устройства
 void          *nciParams;  // Адрес строки параметров
 rdxLine       *line;       // Линия
 rdxUInt32      tagCount[4];  // Количество тэгов в конфигурации
 rdxTag       **tagArray[4];  // Массив тэгов (сортировка по ключу)
 rdxTag        *tagState;     // Тэг состояния устройства (ОТКЛ\ВКЛ, устанавливается протоколом на этапе инициализации)
 rdxUInt32      flags;        // Флаги состояния устройства

 rdxUInt32      fldCount;     // Количество полей НСИ
 rdxUnit       *next;         // Дублированные устройства при резервировании линий
 void          *temp;         // Поле может использоваться протоколом
```

```
};
```

7.7. Структура элемента обобщенных данных

```
struct rdxVariant
{
 rdxTime      DT;        // Время верхнего уровня
 rdxTime      DTCP;      // Время нижнего уровня
 rdxUInt32    FLAG;      // Флаги качества
 rdxVarData   VALUE;     // Данные
 rdxByte      TYPE;      // Тип данных
 rdxByte      cot;       // Причина передачи
};
```

7.8. Структура тега для ретрансляции

```
struct rdxTag{
 rdxUInt32     number;     // Номер
 void         *nciParams;  // Адрес строки параметров
 rdxLine      *line;       // Линия
 rdxUnit      *unit;       // Устройство
 rdxUInt32     flags;      // Флаги состояния тэга
 rdxByte       tagType;    // Тип тэга
 rdxProcParam  *proc;      // Параметры обработки
 rdxUInt32     refCount;   // Количество тэгов ретрансляции
 rdxTag       **refArray;  // Массив тэгов ретрансляции
 rdxVariant    *valueLast; // Последнее значение
 rdxUInt32     valCount;   // Количество элементов в очереди данных
 rdxUInt32     valQueued;  // Количество элементов добавленных в очередь данных
 rdxTag       *next;       // Дублированные тэги при резервировании линий
 void         *temp;       // Поле может использоваться протоколом
};
```

7.9. Дополнительные структуры для работы с тегами

```
struct rdxProcParam // Параметры обработки
{
 rdxUInt32      procType;        // Тип обработки
 rdxProcParamData procParam;   // Параметры
};

union rdxProcParamData  // Параметры обработки (данные)
{
 rdxProcAnalog   analog;   // Аналоговые
 rdxProcDiscrete discrete;  // Дискретные
};

struct rdxProcAnalog  // Параметры обработки аналогового сигнала
{
 rdxDouble      procMul;       // Множитель
 rdxDouble      procAdd;       // Смещение
 rdxDouble      procAperture;  // Апертура
 rdxDouble      procClear;     // Обнуление
};

struct rdxProcDiscrete          // Параметры обработки дискретного сигнала
{
 rdxBool       procInvert;   // Инверсия
 rdxUInt32      procRate;    // Допустимый темп изменения состояния ТС в секундах
 rdxUInt32      procCount;  // Предельное число быстрых изменений ТС
};

union rdxParamData
```

```
{
 rdxBool        vBool;
 rdxInt32       vInt32;
 rdxDouble      vDouble;
 rdxVarBlob     vBlob;
};

struct rdxParamValue
{
 rdxUInt32      TYPE;       // Тип данных
 rdxParamData   VALUE;  // Данные
};
```

ПРИЛОЖЕНИЕ 1

Информационное. Перечень терминов.

Перечень терминов

Сервер Rdx	- сервер ввода-вывода Rdx
ПМ	- протокольный модуль
ОС	- операционная система
ТУ	- телеуправление
ПО	- программное обеспечение
ВУ	- верхний уровень
НСИ	- нормативно-справочная информация
UTC	- Всемирное координированное время

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ									
Изм	Номера листов (страниц)				Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
	измененных	замененных	новых	аннулированных					