

РУКОВОДСТВО ПО РАЗРАБОТКЕ ДРАЙВЕРОВ

Пакет прикладных программ, SCADA система
“КОНТУР II”

КИЕВ 2004

Содержание

1	Введение	3
2	Структура и принцип построения драйвера.....	3
2.1	Функциональные модули драйвера, модуль настройки.....	3
2.1.1	Функция настройки канала	3
2.1.2	Функция настройки устройства.....	4
2.1.3	Функция настройки Тэга	5
2.2	Функциональные модули драйвера, модуль исполнения	6
2.2.1	Функции инициализации.....	6
2.2.2	WriteChannel	7
3	Библиотеки функций	7
3.1	ServerTypes	7
3.2	ServerLibrary	7
3.2.1	Функции автоматического перебора (поиска) тэгов	8
3.2.2	Функции синхронизации (при асинхронном обмене).....	9
3.2.3	Функции, подтверждающие изменение переменной драйвером	9
3.2.4	Функции преобразования.....	9
3.3	Библиотеки работы с последовательным портом.....	10

1 Введение

SCADA система КОНТУР содержит ряд драйверов для подключения множества устройств различного вида. Это драйвера: Carel, Krohne, Modbus, LON, MPI и другие. Однако, видов устройств существует множество и, естественно, не все они представлены в текущем наборе драйверов системы. В этом случае есть три пути решения этой проблемы: 1) использовать предоставляемый с устройствами OPC сервер (если таковой существует) через OPC_DA_Driver (драйвер OPC Data Access) сервера системы КОНТУР; 2) разработать драйвер к системе КОНТУР самостоятельно; 3) обратиться к разработчикам системы КОНТУР, чтобы они реализовали требуемый драйвер.

В данном документе представлена инструкция самостоятельной разработки драйвера разработчиком АСУТП.

2 Структура и принцип построения драйвера

Драйвер – это библиотека DLL со специализированным набором функций.

Функционально драйвер делится на два больших модуля: модуль настройки (Design Time) и модуль исполнения (Run Time).

Структурно драйвер содержит модули: Канал, Устройство, Тэг (технологическая переменная).

2.1 Функциональные модули драйвера, модуль настройки

Модуль настройки – это функции отображения диалоговых окон настройки канала, устройства и тэга. Он включает функции: ChannelSettings, DeviceSettings, TagSettings.

2.1.1 Функция настройки канала

Формат функции настройки канала:

```
EXPORTFUNC int ChannelSettings
```

```
(  
    PChannelProperties CP,  
    int CurrentLanguage  
);
```

Как видно, в функцию ChannelSettings передаётся указатель на настройки канала и значение (код) текущего языка (Русский - 1049, Английский – 2057, и т.д.), который был выбран при установке системы.

Структура настроек канала имеет следующий формат:

```
struct TChannelProperties {  
    wchar_t* Name;           // Имя канала  
    wchar_t* Driver;        // Название драйвера  
  
    int Frequency;         // Период опроса канала (в миллисекундах)  
  
    wchar_t* P1;           // Пользовательская настройка, тестовый формат "WideChar"  
    wchar_t* P2;  
    wchar_t* P3;  
    wchar_t* P4;  
    wchar_t* P5;  
  
    real P6;               // Пользовательская настройка, формат с плавающей точкой (8 байт)  
    real P7;  
    real P8;  
    real P9;  
    real P10;  
    real P11;  
    real P12;
```

```
real P13;  
real P14;  
real P15;  
};
```

В этой структуре, присутствуют специализированные настройки (Имя канала, название драйвера, период опроса), а также – пользовательские. Назначение пользовательских настроек определяются разработчиком драйвера.

Пример функции настройки канала драйвера можно увидеть в файле «Main.cpp», шаблона драйвера, предоставляемого разработчиками системы КОНТУР. Если Вы используете шаблон для разработки собственного драйвера, то Вам необязательно вникать в принцип работы этой функции и менять что-либо в её работе.

Однако, при написании собственного драйвера, окно настройки канала может отличаться от шаблона. Тогда необходимо отредактировать ресурс диалога «IDD_CHANNEL_DLG», а также код функций “Init”, “ DoDataExchange”, “ InputSettings”, “OnBnClickedChannelOk”, функции обработки событий диалога класса “TChannelDlg” (в файле “TChannel.cpp”), чтобы обработать изменения в настройках, внесённые пользователем и передать их в сервер.

2.1.2 Функция настройки устройства

Формат функции настройки устройства:

```
EXPORTFUNC int DeviceSettings  
(  
    PDeviceProperties DP, // Настройки устройства  
    PChannelProperties CP, // Настройки канала  
    int CurrentLanguage, // Язык  
    PDevicePropertiesArray pDPArray, // Массив устройств  
    int* piCount // Количество устройств в массиве  
);
```

Как видно, в функцию DeviceSettings передаётся указатель на настройки устройства, канала, значение (код) текущего языка (Русский - 1049, Английский – 2057, и т.д.), который был выбран при инсталляции системы, а также указатель на массив устройств и указатель на количество устройств. Последние две настройки предназначены для драйверов, в которых реализован, например, автоматический поиск устройств и все настройки найденных устройств передаются в сервер.

Структура настроек устройства имеет следующий формат:

```
struct TDeviceProperties {  
    wchar_t* Name; // Имя устройства  
    int Adress; // Адрес устройства  
  
    wchar_t* P1; // Пользовательские текстовые настройки для устройства  
    wchar_t* P2;  
    wchar_t* P3;  
    wchar_t* P4;  
    wchar_t* P5;  
  
    real P6; // Пользовательские числовые настройки для устройства  
    real P7;  
    real P8;  
    real P9;  
    real P10;  
    real P11;  
    real P12;  
    real P13;  
    real P14;
```

```
real P15;  
};
```

В этой структуре, присутствуют специализированные настройки и пользовательские. Назначение пользовательских настроек определяются разработчиком драйвера.

Пример функции настройки устройства драйвера можно увидеть в файле «Main.cpp», шаблона драйвера, предоставляемого разработчиками системы КОНТУР. Если Вы используете шаблон для разработки собственного драйвера, то Вам необязательно вникать в принцип работы этой функции и менять что-либо в теле этой функции.

Однако, при написании собственного драйвера, окно настройки устройства может отличаться от шаблона. Тогда необходимо отредактировать ресурс диалога «IDD_DEVICE_DLG», а также код функций “Init”, “ DoDataExchange”, “ InputSettings”, “OnBnClickedDeviceOk”, функции обработки событий диалога класса “TDeviceDlg” (в файле “TDevice.cpp”), чтобы обработать изменения в настройках, внесённые пользователем и передать их в сервер.

2.1.3 Функция настройки Тэга

Формат функции настройки Тэга:

```
EXPORTFUNC int TagSettings  
(  
    PPhysicTagProperties TP, //Настройки Тэга  
    PDeviceProperties DP, // Настройки устройства  
    PChannelProperties CP, // Настройки канала  
    int CurrentLanguage, // Язык  
    PPhysicTagPropertiesArray pTPArray, // Множество тэгов  
    int* piCount // Количество тэгов  
);
```

Как видно, в функцию TagSettings передаётся указатель на настройки устройства, канала, тэга, значение (код) текущего языка (Русский - 1049, Английский – 2057, и т.д.), который был выбран при инсталляции системы, а также указатель на массив тэгов и указатель на количество тэгов. Последние две настройки предназначены для драйверов, в которых реализован, например, автоматический поиск тэгов устройства и все настройки найденных тэгов передаются в сервер (например, так реализован драйвер OPC DataAccess).

Структура настроек тэга имеет следующий формат:

```
struct TPhysicTagProperties {  
    wchar_t* Name; // Имя Тэга  
    wchar_t* Legend; // Легенда (описание)  
    wchar_t* Units; // Размерность  
  
    real Adress; // Адрес  
  
    int VType; // Тип принимает значения по типу Variant  
    int ScaleType; // Масштабирование  
    int SignDigits; // Число цифр после запятой  
    int ForegroundColor; // Цвет надписи  
    int BackGroundColor; // Цвет фона  
  
    real HighEU; // Максимальный предел переменной до масштабирования  
    real LowEU; // Минимальный предел переменной до масштабирования  
    real HighIR; // Максимальный предел переменной после масштабирования  
    real LowIR; // Минимальный предел переменной до масштабирования  
    real DeadBand; // Мёртвая зона переменной (фильтрация)  
  
    real LowLowAlarm; // Нижний аварийный предел переменной
```

```
real LowAlarm; // Нижний предаварийный предел переменной
real HiAlarm; // Верхний предаварийный предел переменной
real HiHiAlarm; // Верхний аварийный предел переменной

real P1; // Пользовательские числовые настройки
real P2;
real P3;
real P4;
real P5;

wchar_t* P6; // Пользовательские текстовые настройки
wchar_t* P7;
```

```
BOOL HistorySave; //Архивировать историю изменений тэга или нет
```

```
};
```

В этой структуре, присутствуют специализированные настройки и пользовательские. Назначение пользовательских настроек определяются разработчиком драйвера. Специализированные настройки – это настройки, которые сервер будет использовать для архивирования, масштабирования, формирования аварийных сообщений, фильтрации.

Пример функции настройки устройства драйвера можно увидеть в файле «Main.cpp», шаблона драйвера, предоставляемого разработчиками системы КОНТУР. Если Вы используете шаблон для разработки собственного драйвера, то Вам необязательно вникать в принцип работы этой функции и менять что-либо в теле этой функции.

При написании собственного драйвера, окно настройки тэга может отличаться от шаблона. Тогда необходимо отредактировать ресурс диалогов «IDD_TAGS_DLG», а также код функций “Init”, “ DoDataExchange”, “ InputSettings”, функции обработки событий диалога класса “TagsPP” (в файле “TagsPP.cpp”), чтобы обработать изменения в настройках, внесённые пользователем и передать их в сервер.

2.2 Функциональные модули драйвера, модуль исполнения

Модуль исполнения содержит следующие функции: InitChannel, CloseChannel, InitDevice, CloseDevice, InitTag, CloseTag, WriteChannel.

2.2.1 Функции инициализации

InitChannel, CloseChannel вызываются при инициализации канала и его закрытии. В эти функции передаётся указатель на настройки канала, т.е. здесь доступны все те же настройки, что и в ChannelSettings. Здесь можно инициализировать определённые настройки канала для всего времени его работы.

InitDevice, CloseDevice вызываются при инициализации устройства и его закрытии. В эти функции передаётся указатель на настройки устройства, т.е. здесь доступны все те же настройки, что и в DeviceSettings. Также здесь доступны настройки канала таким способом, как в данном примере доступна пользовательская настройка канала “P6”.

```
((PChannel)Device->Channel)-> P6 = 5;
```

InitTag, CloseTag вызываются при инициализации тэга и его закрытии. В эти функции передаётся указатель на настройки тэга, т.е. здесь доступны все те же настройки, что и в TagSettings. Также здесь доступны настройки устройства и канала, например:

```
PDevice(Tag->Device)->P8 = 5;
```

```
PChannel(PDevice(Tag->Device)->Channel)->P6 = 5;
```

Эти конструкции немного громоздки, но тем не менее, в конечном итоге, упрощают работу программиста (не нужно делать лишних вызовов и обработок функции, передавать дополнительные параметры) и придают системе гибкость.

2.2.2 WriteChannel

Эту функцию можно назвать «сердцем» работы драйвера. Именно при помощи этой функции сервер обменивается данными с драйвером в “Run Time”.

Эта функция имеет формат:

```
EXPORTFUNC int WriteChannel  
(  
    PChannel      Channel,  
    int           CurrentLanguage,  
    wchar_t*     wcsErrorMessage,  
    DWORD*       pTagChangedStruct,  
    int*         piCountChanged  
);
```

В функцию передаётся указатель на весь массив устройств и тэгов через единственный указатель “Channel”. Также в функцию передаётся код языка, а возвращает она серверу сообщения об ошибках (если таковые имеются) и массив указателей на изменённые тэги (для ускорения обработки).

Для работы с функцией **WriteChannel** рекомендуем использовать библиотеку «ServerLibrary.h».

Принципиально, в теле функции, необходимо перебрать все тэги, чтобы определить, были ли они изменены пользователем, и сформировать посылку в контроллер. Затем следует снова перебрать все тэги, чтобы вернуть в сервер реальные значения тэгов, полученные с контроллера. Также возможны вариации, когда есть тэги только на запись, только на чтение, реализован асинхронный обмен и т.д.

Значение переменной находится в элементе “IValue” (см. пример в шаблоне).

3 Библиотеки функций

Для разработки драйверов с Контуром предлагаются следующие библиотеки:

```
“ExternalContainerUnit.h”;  
“ServerLibrary.h”;  
“ServerTypes.h”;  
“SerialComm.h”, “SerialComm.cpp”.
```

3.1 ServerTypes

Эта библиотека – набор типов данных. Эти типы Вы используете, когда работаете с настройками в “Run Time”. Это – настройки канала, устройства, тэга.

Всё что Вам необходимо знать про данную библиотеку, это то, что со временем число и функциональность настроек системы может расширяться. Тогда, чтобы использовать новые настройки и функции Вам потребуется использовать новую версию библиотеки.

3.2 ServerLibrary

Данная библиотека содержит набор функций для упрощения обработки данных в функции WriteChannel. Перечислим эти функции:

```
PDevice FindFirstDevice(PChannel Channel)  
PTagPhysic FindFirstReadTag(PDevice Device)  
PTagPhysic FindFirstWriteTag(PDevice Device)  
PTagPhysic FindFirstTag(PDevice Device)  
PTagPhysic FindFirstTagNoWriteIt(PDevice Device)
```

```
PDevice FindNextDevice(PChannel Channel)  
PTagPhysic FindNextReadTag(PDevice Device)
```

```
PtagPhysic FindNextWriteTag(PDevice Device)
PtagPhysic FindNextTag(PDevice Device)
PtagPhysic FindNextTagNoWriteIt(PDevice Device)

bool UseDevice(PDevice Device)
void EndUseDevice(PDevice Device)
bool UseReadTag(PtagPhysic Tag) //Only for reading from controller
void EndUseReadTag(PtagPhysic Tag)
bool UseTag(PtagPhysic Tag) // Only for writing to controller
void EndUseTag(PtagPhysic Tag)

void SetTagError(PtagPhysic Tag, DWORD* TagsChanged, int* CountChanged)
void SetTagOK(PtagPhysic Tag, DWORD* TagsChanged, int* CountChanged)

void ValToByte(BYTE* dest, void* source, int size, int startbyte) //IEEE-754
void ByteToVal(void* dest, BYTE* source, int size, int startbyte) //IEEE-754
```

Рассмотрим каждую функцию (примеры использования некоторых функций есть в «Шаблоне»):

3.2.1 Функции автоматического перебора (поиска) тэгов

FindFirstDevice - выполняет поиск первого устройства в канале, возвращает NULL, если канал не содержит устройств;

FindFirstReadTag – выполняет поиск первого тэга, который не был изменён и не требует записи в контроллер, возвращает NULL, если устройство не содержит тэгов;

FindFirstWriteTag – выполняет поиск первого тэга, который был изменён и требует запись в контроллер, возвращает NULL, если устройство не содержит тэгов;

FindFirstTag – выполняет поиск любого (независимо на чтение или на запись) первого тэга, возвращает NULL, если устройство не содержит тэгов;

FindFirstTagNoWriteIt – выполняет поиск любого (независимо на чтение или на запись) первого тэга, возвращает NULL, если устройство не содержит тэгов;

FindNextDevice - выполняет поиск следующего устройства в канале, возвращает NULL, если устройство не найдено;

FindNextReadTag – выполняет поиск следующего тэга, который не был изменён и не требует записи в контроллер, возвращает NULL, если тэг не найден;

FindNextWriteTag – выполняет поиск следующего тэга, который был изменён и требует запись в контроллер (определяет по флагу сигнала записи), возвращает NULL, если тэг не найден (сбрасывает флаг, сигнализирующий о том, что тэг был записан);

FindNextTag – выполняет поиск любого (независимо на чтение или на запись) следующего тэга, возвращает NULL, если тэг не найден (сбрасывает флаг, сигнализирующий о том, что тэг был записан)

FindNextTagNoWriteIt – выполняет поиск любого (независимо на чтение или на запись) следующего тэга, возвращает NULL, если тэг не найден (не сбрасывает флаг, сигнализирующий о том, что тэг был записан). Используйте эту функцию, если хотите самостоятельно проверять и управлять флагом сигнала записи (Tag->WriteIt). Также помните, что для изменения Тэга в сервере с использованием этой функции, нужно использовать функцию UseTag

3.2.2 Функции синхронизации (при асинхронном обмене)

Эти функции позволяют синхронизировать работу сервера и драйвера, если драйвер реализован с асинхронным обменом, т.е. контроллер или другое устройство инициирует передачу данных самостоятельно.

UseDevice – сообщает драйверу может ли он использовать устройство или его тэги для изменения параметров или значений и «забирает» его для драйвера. Возвращает TRUE, если устройство свободно, FALSE, если занято. Здесь, при необходимости, можно подождать, пока сервер освободит устройство для записи. Обычно это выглядит так:

```
While !(UseDevice(Device)) do Sleep(1);
```

EndUseDevice – освобождает устройство от использования драйвером. Обязательно вызывать после UseDevice!!!

UseReadTag – сообщает драйверу может ли он использовать тэг для изменения его параметров или значений (т.е., изменения в сервере, чтения с контроллера) и «забирает» его для драйвера. Возвращает TRUE, если тэг свободен, FALSE, если занят.

EndUseReadTag – освобождает тэг от использования драйвером. Обязательно вызывать после UseReadTag!!!

UseTag – сообщает драйверу нужно ли использовать тэг для записи его параметров или значений в контроллер, т.е. проверяет флаг WriteIt. Возвращает TRUE, если тэг был изменён клиентом и его нужно записать в контроллер, FALSE, если нет.

EndUseTag – освобождает тэг от использования драйвером. Обязательно вызывать после UseTag!!!

3.2.3 Функции, подтверждающие изменение переменной драйвером

SetTagError – эта функция подтверждает изменение тэга (дополняет массив pTagChangedStruct, изменившихся тэгов в функции WriteChannel) и устанавливает негативное качество переменной (значение переменной не было принято с контроллера, произошла ошибка контрольной суммы и т.д.). Вообще качество переменной можно варьировать.

SetTagOK – эта функция подтверждает изменение тэга (дополняет массив pTagChangedStruct, изменившихся тэгов в функции WriteChannel) и устанавливает позитивное качество переменной (192).

3.2.4 Функции преобразования

ValToByte – преобразовывает значение переменной в последовательность байт для записи в последовательный канал данных. Здесь:

```
BYTE* dest – указатель на массив байт  
void* source – указатель на переменную – источник  
int size – размер переменной в байтах  
int startbyte – номер начального байта из массива
```

ByteToVal – преобразовывает последовательность байт в значение переменной. Здесь:

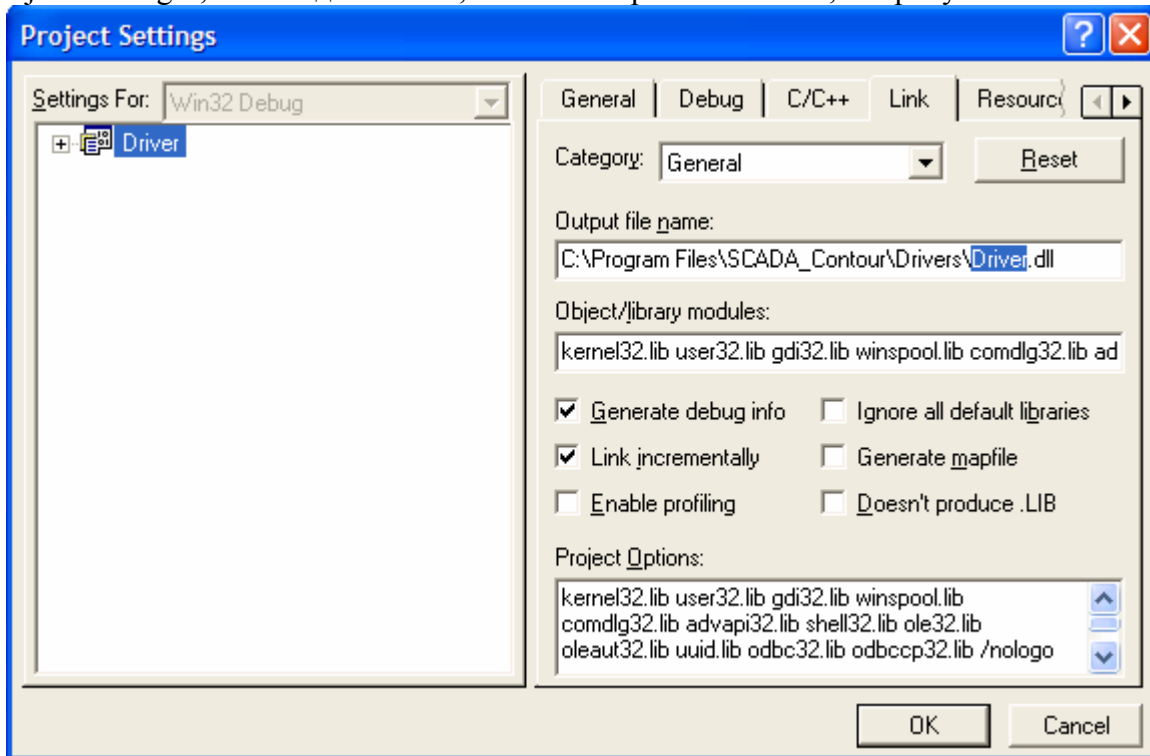
```
void * dest – указатель на переменную – источник  
BYTE * source – указатель на массив байт  
int size – размер переменной в байтах  
int startbyte – номер начального байта из массива
```

3.3 Библиотеки работы с последовательным портом

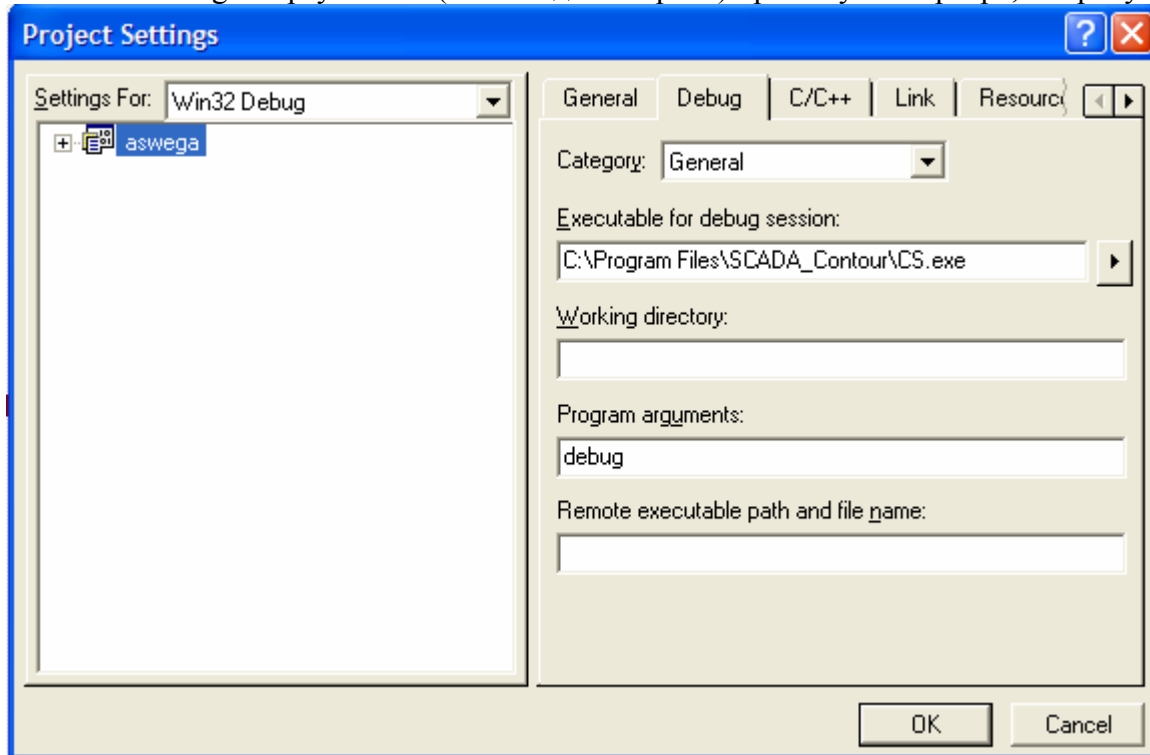
“SerialComm.h”, “SerialComm.cpp” – здесь содержится класс для работы с последовательным портом. Описание этой библиотеки будет дано позднее.

4 Компиляция и отладка

Чтобы откомпилировать драйвер в Microsoft Visual Studio 6.0, нужно установить имя файла и папку. Папка – это поддиректория “Drivers” папки, в которую была проинсталлирована система КОНТУР. Имя файла – это будет имя Вашего драйвера. Его Вы можете задать самостоятельно. Имя файла и папка для компиляции устанавливаются в меню “Project\Settings”, на вкладке “Link”, в поле “Output File Name”, см. рисунок.



Для отладки драйвера нужно задать имя сервера в том же меню, на вкладке “Debug”, а также ключ “debug” в аргументах (в командной строке) при запуске сервера, см. рисунок.



Здесь важно понимать, что аргумент “debug” сервера переводит его в режим отладки. В этом режиме использование нескольких каналов с одним драйвером будет приводить к

некорректной работе этого драйвера. Т.е. отлаживать драйвер советуем используя один канал в проекте.