Программное обеспечение микропроцессорной системы управления

OPC-сервер МПСУ

Руководство программиста

ЯАЦВ.00045-01 33 06

Листов …

2003

Аннотация

Данный документ является руководством программиста и предназначен для использования при создании различных программно-технических комплексов и систем, основанных на взаимодействии «верхнего» уровня на базе IBM PC совместимых компьютеров с «нижним» уровнем микропроцессорных контроллеров на базе промышленного контроллера МПСУ. В настоящем руководстве приведены общие сведения по технологии OPC, сведения о структуре системы взаимодействия «верхнего» уровня с МПСУ через OPC-сервер, о реализации модулей системы. Применение настоящего руководства позволяет практически использовать OPC-сервер МПСУ при разработке систем типа АСУТП.

В результате разработки OPC-сервера МПСУ была достигнута цель организации эффективного стандартного доступа из различных SACADA систем и других средств программирования интерфейсов «верхнего» уровня к каналам связи с МПСУ.

Следует уточнить, что название «OPC-сервер МПСУ» является принадлежностью промышленного контроллера МПСУ на безе процессорных модулей типа М231, М251…М255. Это весьма существенно, поскольку предполагает наличие встроенного программного обеспечения контроллера, включающего в себя программу «супервизор», а также обмен МПСУ с «верхним» уровнем по последовательным и параллельным каналам по «протоколу обмена МПСУ». Подробное описание «протокола обмена МПСУ» приведено в руководстве программиста ЯАЦВ.00045-01 33 05. Уточним, что в настоящем руководстве применяются такие названия, как «система удалённого программирования» (система программирования МПСУ с «верхнего» уровня на основе «протокола обмена МПСУ»); «супервизор» (принятое для системы удаленного программирования МПСУ название программного обеспечения, которое располагается в ПЗУ контроллера МПСУ и обеспечивает конфигурирование и стартовую диагностику МПСУ, взаимодействие с компьютером «верхнего» уровня по каналам связи по протоколу обмена МПСУ, выполнение функций управления УСО, принятых от компьютера «верхнего» уровня).

Содержание

1 Введение

1.1 При создании различных автоматизированных систем в области контроля и управления технологическими процессами применяется большое многообразие программных средств, позволяющих разработать сравнительно в сжатые сроки интерфейсы пользователя на «верхнем» уровне. В частности, к таким средствам относятся SCADA-системы (Supervisory for Control And Data Acquisition / Система верхнего уровня для управления и сбора данных). Особый интерес вызывает организация взаимодействия между SCADA-системой и уровнем контроллеров, которые собственно осуществляют непосредственный ввод сигналов и вывод управляющих воздействий в системе автоматизации. Это взаимодействие должно быть эффективным с точки зрения сопряжения функций контроллера с переменными (входными и выходными) в SCADA системе. Под эффективностью можно понимать также так называемое время отклика, то есть, например, для входного сигнала – это задержка между моментом изменения сигнала в контроллере до реакции на это изменение в SCADA-системе. Это взаимодействие должно быть стандартным, чтобы практически не зависимо от вида применяемой SCADA-системы правила работы с контроллером не менялись.

1.2 Доступ к уровню контроллеров (не зависимо от физических каналов связи с ними) может производиться через специальные драйверы, которые по отношению к SCADA –системе могут быть «встроенные» и внешние. В любом случае, они являются с одной стороны принадлежностью контроллера (протокола обмена с контроллером), а с другой стороны являются принадлежностью конкретной SCADA-системы. Для МПСУ, в частности, разработан драйвер обмена с системой «Круг-2000/NT», а также имеется внешний драйвер для работы с системой «Трейс Моуд 4.1x. Но для любой другой SCADA-системы потребовался бы свой драйвер. То есть решение проблемы взаимодействия по данному пути каждый раз возлагается или на производителя контроллера, или на разработчика SCADA-системы (последнее для МПСУ очень затруднительно). При этом драйверы должны обладать конкретными интерфейсами и функциями такими, чтобы программы «верхнего» уровня различных производителей могли их без проблем использовать.

1.3 Большое количество программ «верхнего» уровня в указанной области реализуются в настоящее время на базе персональных компьютеров под операционными системами Microsoft Windows (Windows 9x, Windows NT/2000/XP). Количество пользовательских программ продолжает расти. Для создания на базе таких стандартов операционной системы Windows, как OLE (Object Linking and Embedding), COM (Component Object Model) и DCOM (Distributed Component Object Model) единой процедуры доступа и единого интерфейса между пользовательскими программами с одной стороны и программируемыми контроллерами с другой стороны была предложена технология OPC. Технология или стандарт OPC (OLE for Process Control) является унифицированным средством, при помощи которого компоненты (контроллеры) различных производителей в рамках некоторой системы автоматизированного управления могли бы связываться с программой «верхнего» уровня по стандартизованному интерфейсу. В современной редакции спецификаций OPC описываются процедуры чтения и записи между программным обеспечением и компонентами систем автоматизированного управления производством, функции обработки сообщений, обеспечения безопасности данных и доступа к данным архивов.

1.4 OLE for Process Control (OPC) в качестве стандартной процедуры обеспечения всестороннего доступа к данным уровня контроллеров дает следующие преимущества:

- Производители контроллеров должны поставлять разработчикам системы АСУТП, базирующихся на самых разных программных средствах «верхнего» уровня

(поддерживающих, естественно, технологию ОРС) только некоторый набор стандартизованных программных средств (ОРС-сервер), нет необходимости каждый раз разрабатывать новый драйвер;

•Для разработчиков SCADA-систем также отпадает необходимость написания новых драйверов для новых контроллеров («встраивания» контроллеров в свою систему), а также дорабатывать драйвера, если вследствие модернизации некоторого контроллера изменяется набор функций доступа к нему.

•Разработчики АСУТП получают большую свободу выбора при конфигурировании и подборе аппаратных средств решения их задач автоматизации.

1.5 Применительно к МПСУ разработанный ОРС-сервер позволяет предложить применение ПК МПСУ для включения в проекты разработки систем АСУТП, в которых для «верхнего» уровня может быть выбрана практически любая современная SCADA-система. Единственным условием совместимости с программной точки зрения является поддержка технологии ОРС SCADA-системой.

1.6 Договоримся о названиях и обозначениях в последующих пунктах настоящего руководства, чтобы не загромождать текст и обеспечить однозначность понимания.

▪ контроллер МПСУ, ПК МПСУ - МПСУ
▪ программа или система «верхнего уровня», SCADA-система - приложение или клиент
▪ технология, стандарт ОРС - ОРС
▪ автоматизированная система, АСУТП - система

вернуться на **“Содержание”**

2 Краткий обзор спецификации ОРС

2.1 Спецификация ОРС определяет описание COM интерфейсов и COM объектов, при этом их реализация не регламентируется, но ожидаемое поведение этих интерфейсов и объектов специализируется. В настоящее время имеются следующие ОРС стандарты:

1) OPC Common Definitions and Interfaces – общие для всех спецификаций интерфейсы

2) Data Access Custom Interface Standard – спецификация COM-интерфейсов для обмена оперативными данными, программирование на C++

3) Data Access Automation Interface Standard – спецификация COM-интерфейсов для обмена оперативными данными, программирование на языках, поддерживающих Автоматизацию(Automation), например, Microsoft Visual Basic.

4) OPC Batch Custom Interface Specification – спецификация COM–интерфейсов конфигурирования оборудования, программирование на C++

5) OPC Batch Automation Interface Specification – спецификация COM–интерфейсов конфигурирования оборудования, программирование на языках программирования, поддерживающих Автоматизацию (Automation).

6) OPC Alarms and Events Interface Specification – спецификация COM–интерфейсов для обслуживания событий (event) и нештатных ситуаций (alarm), программирование на C++

7) Historical Data Access Custom Interface Standard – спецификация COM-интерфейсов для работы с хранилищами данных, программирование на C++

8) OPC Security Custom Interface – спецификация COM–интерфейсов для обработки прав доступа к данным, программирование на C++

Вышеперечисленные спецификации призваны охватить все аспекты взаимодействия с контроллерным оборудованием.

Спецификация OPC подразумевает архитектуру построения системы типа «клиент-сервер». OPC–клиент (клиенты) работает с данными, предоставляемыми OPC-сервером (серверами). Взаимодействие программы-клиента и OPC-сервера осуществляется прежде всего в контексте технологии COM, то есть посредством интерфейсов.

Спецификация OPC определяет открытый интерфейс, который делает возможным создание стандартизованных процедур обмена данными между приложениями (SCADA, DSC системы) и аппаратурой (контроллерами).

В общем случае, структуру системы, построенной по стандарту OPC, можно представить в виде (рис.1)

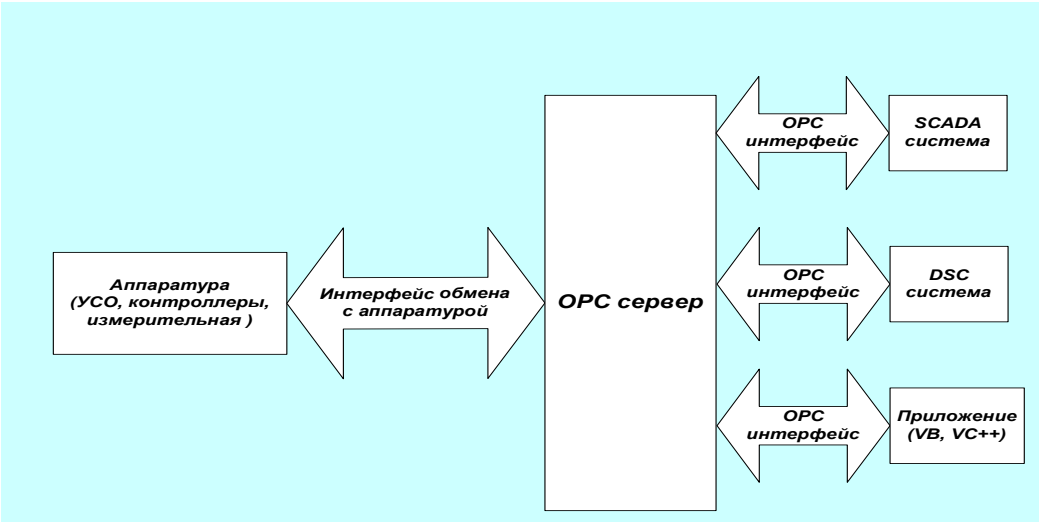

Рис.1 Структурная схема системы с применением OPC решения

OPC-сервер предоставляет в распоряжение OPC-клиента регламентированные спецификацией COM-интерфейсы, инкапсулируя всю специфику обмена данными с оборудованием. Производитель OPC–сервера разрабатывает программный код, определяющий устройства и данные, к которым OPC–сервер имеет доступ, кроме того, задаёт имена данных и детали физического доступа к этим данным.

На верхнем уровне OPC–сервер включает в себя несколько объектов: сервер, группу и единицу данных. OPC–сервер объект оперирует информацией о сервере и служит контейнером для объектов OPC–групп. Объекты OPC–групп оперируют информацией о себе и обеспечивают механизмы для хранения и логической организации OPC–переменных.

2.2 Переменные

Переменная может быть любого типа, допустимого в OLE: различные целые и вещественные типы, логический тип, строковый, дата, валюта, вариантный (VARUANT) и так далее. Кроме того, переменная может быть массивом.

Каждая переменная обладает свойствами. Различаются обязательные свойства, рекомендуемые и пользовательские. Обязательными свойствами должна обладать каждая переменная. Это, во-первых, текущее значение переменной, тип переменной и права доступа (чтение/запись). Во-вторых, очень важные свойства – качество переменной и метка времени. Технология OPC ориентирована на работу с оборудованием, которое, в принципе,

может давать сбои, и по этой причине есть вероятность получения не корректных значений переменных в ОРС-сервере, о чем и уведомляется клиент через качество переменной (*хорошее/ плохое/ неопределенное*). Метка времени сообщает о том, когда данная переменная получила определённое значение и/или качество.

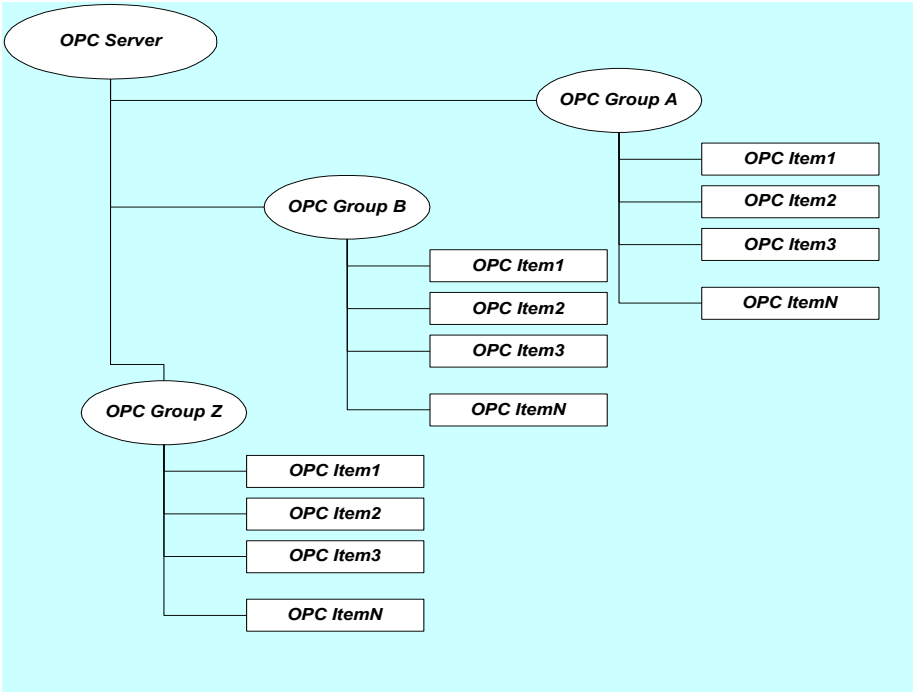

Рис.2 Основные элементы ОРС сервера

Еще одним важным свойством переменной является частота её обновления, то есть частота, с которой обновляется значение и качество переменной в результате опроса оборудования. Таким образом, ОРС–переменная является элементом данных, которыми выполняется обмен с устройствами.

2.3 Группы

ОРС группа (OPC Group) предназначена для логической организации данных в ОРС сервере. ОРС группа служит контейнером для ОРС переменных и других ОРС групп. Таким образом, структура данных в ОРС сервере имеет иерархическую структуру (см. Рисунок 2.).

2.4 Получение данных ОРС клиентом от ОРС сервера, запись данных ОРС клиента в ОРС сервер

Существует три основных способа получения данных ОРС-клиентом от ОРС-сервера: синхронное чтение, асинхронное чтение и подписка. При синхронном чтении клиент посылает запрос серверу с группой интересующих его переменных и ждет, когда сервер его выполнит. При асинхронном чтении клиент посылает запрос серверу, а сам продолжает работать, при выполнении запроса клиент получает уведомление. В случае подписки клиент передает серверу группу интересующих его переменных, а сервер, затем регулярно присылает клиенту информацию об изменившихся переменных из этой группы. Каждый клиент может одновременно поддерживать несколько групп с разной скоростью обновления.

Запись данных в ОРС-сервер выполняется аналогично, только при этом нет записи по подписке, поскольку обновление выходных переменных осуществляет сам клиент.

2.5 Источники данных

Спецификация ОРС регламентирует лишь интерфейс между ОРС клиентами и ОРС серверами. И она абсолютно не регламентирует способ получения этих данных от оборудования. Разработчик сам решает, где и как их брать.

2.6 Адресация данных

Адрес переменной в пространстве имен ОРС сервера называется ***тегом*** (tag). *Теги являются уникальными идентификаторами переменных в пространстве имен ОРС сервера*. Тег отражает принадлежность ОРС-переменной той или иной группе (группам), например, ***UniOPCAuto.Device1.unit1.Chanel***.

вернуться на **“Содержание”**

3 Свойства ОРС-сервера МПСУ

3.1 Назначение ОРС-сервера МПСУ

ОРС-сервер МПСУ предназначен для обеспечения возможности обмена данными между МПСУ и приложениями в соответствии со спецификациями OPC Data Access Custom Interface Standard и OPC Data Access Automation Interface Standard.

ОРС сервер МПСУ отвечает следующим требованиям:

1) Обеспечивает реализацию полного набора COM интерфейсов и COM объектов, регламентированных спецификациями OPC Data Access Custom Interface Standard и OPC Data Access Automation Interface Standard.

2) Обеспечивает возможность взаимодействия приложений с МПСУ посредством открытого стандарта ОРС. При этом специфика обмена данными с МПСУ скрыта от ОРС клиента (клиентов).

3) Предоставляет в распоряжение ОРС клиента набор функций МПСУ, достаточный для реализации обработки различных переменных, для работы с основными модулями УСО.

4) ОРС-сервер автоматически конфигурирует пространство тегов, предоставляемое в распоряжение ОРС клиентов в соответствии с текущей конфигурацией МПСУ (составом модулей УСО). Какие-либо изменения состава модулей УСО, входящих в МПСУ, автоматически отражаются на пространстве тегов ОРС сервера.

5) Обеспечивает надежность обмена данными с МПСУ, осуществляет контроль над обменом данными и информирует ОРС клиента о возникающих в процессе обмена ошибках.

6) В плане быстродействия системы ОРС-сервер не вносит каких-либо существенных задержек в процесс обмена данными. Скорость обмена данными приложения (ОРС клиента) с МПСУ в основном ограничивается лишь характеристиками приложения, являющегося ОРС клиентом, а также свойствами физического канала связи МПСУ с компьютером (последовательные порты COM1-COM4, параллельные порты).

вернуться на **“Содержание”**

3.2 Источник разработки ОРС-сервера МПСУ

3.2.1 Разработка ОРС-сервера МПСУ выполнена с применением универсального ОРС сервера **Fastwell UniOPC** компании Fastwell. Компания предлагает разработчикам оборудования для АСУТП (контроллеров), так называемый **универсальный ОРС сервер**. В процессе разработки потребовалось реализовать динамическую библиотеку доступа к

данным dataserv.dll, которая обеспечивает специфику обмена с МПСУ по каналам связи. Обмен данными МПСУ с ОРС сервером реализован с использованием «Протокола обмена МПСУ» Базовый исходный код этой библиотеки поставляется в комплекте с **Fastwell UniOPC**. Кроме того, разработана программа, в которой реализованы механизмы конфигурирования ОРС сервера. Обеспечена также синхронизация чтения/записи данных ОРС клиентом.

Использованный способ разработки ОРС-сервера имеет ряд положительных свойств, а именно:

1) Отпадает необходимость в подробном изучении спецификаций, приобретении программных компонентов ОРС Foundation, в полной реализации COM-интерфейсов ОРС сервера. ***Fastwell UniOPC*** в полной степени соответствует спецификациям *OPC Data Access Custom Interface Standard* и *OPC Automation Interface Standard*.

2) Появляется возможность в полной степени реализовать функциональность МПСУ, отразить структуру конфигурации МПСУ в пространстве имен ОРС сервера.

3) Появляется возможность реализовать три варианта обмена данных ОРС клиента с МПСУ (синхронный, асинхронный и подписка)

4) Существенно снижается трудоемкость проекта.

5) Существенно снижается себестоимость проекта, поскольку **Fastwell UniOPC** для целей разработки распространяется бесплатно (для коммерческого использования полученного программного продукта необходимо приобрести лицензию у компании **Fastwell)**.

Другие варианты разработки ОРС-сервера (написание «с нуля», написание с помощью одного из инструментальных пакетов, так называемых *toolkit,* библиотек, реализующих ОРС объекты) являются более трудоёмкими и дорогими.

.

3.2.2 Решена задача реализации обмена ОРС-сервера с МПСУ, для этого динамически связываемая библиотека доступа к данным ***dataserv.dll*** приведена в соответствие со спецификой обмена с МПСУ.

Каркас библиотеки ***dataserv.dll*** включает в себя определение структур тегов, функций для их изменения, а также предоставляет возможность автоматического конфигурирования пространства тегов ОРС сервера – при загрузке библиотеки в адресное пространство ОРС сервера в EntryPoint функции библиотеки можно разместить программный код, выполняющий эту операцию. Интерфейс с библиотекой доступа к данным представлен в **Приложении А Интерфейс с библиотекой доступа к данным dataserv.dll.**

вернуться на **“Содержание”**

3.3 Среда разработки и исполнения

3.3.1 В качестве инструментальной среды разработки была выбрана интегрированная среда разработки *Microsoft Visual C++6 Enterprise Edition*, входящая в пакет *Microsoft Visaul Studio 98*. Выбор обусловлен прежде всего тем, что каркас библиотеки *dataserv.dll* реализован на ЯВУ C++. Перекодирование его на какой-либо иной ЯВУ потребовало бы дополнительного времени. Кроме того, соглашение об именах в таблицах экспорта/импорта данных и функций библиотеки *dataserv.dll,* полностью соответствует соглашению об именах таблиц экспорта/импорта библиотек, генерируемых компилятором *IDE Microsoft Visual C++ 6.* Использование другого компилятора, входящего в состав какой-либо другой среды разработки, повлекло бы за собой необходимость согласования формата имен экспортируемых/импортируемых данных и функций библиотеки *dataserv.dll*.

Следует отметить также то, что использовано применение приемов ООП (объектно-ориеБнтированного программирования), а выбранная среда разработки предлагает набор достаточно мощных инструментов для применения ООП.

Кроме того, быстродействие приложений, разрабатываемых с применением Microsoft Visual C++ 6, достаточно высоко, благодаря встроенным средствам оптимизации генерируемого машинного кода.

В качестве еще одной причины, по которой была *выбрана IDE Microsoft Visual Studio 98*, стоит указать на то, что проектируемая система является Win32 приложением. *Microsoft Visual C++ 6*, благодаря поддержке Win32 API, позволяет в полной степени использовать такие возможности платформы Win32 как многозадачность, многопоточность, сериализация объектов операционной системы, предоставляет практически неограниченные возможности для эффективной работы с памятью, ресурсами процессора.

3.3.2 Операционная система

В качестве операционной системы выбрана операционная среда Microsoft Windows NT 4.0. Данный выбор обусловлен следующими факторами:

1) OPC технология базируется на технологии OLE. Данные технологии реализованы и встроены в операционные системы семейства Microsoft Windows, начиная с Microsoft Windows 95. Windows NT поддерживает COM/DCOM, OLE , а следовательно, в этой среде возможна реализация OPC сервера.

2) Из семейства операционных систем Microsoft Windows, операционная система Microsoft Windows NT является наиболее надежной в плане сетевой безопасности (при работе с DCOM обмен между OPC клиентом и OPC сервером происходит по сети), устойчивости работы, рационального управления ресурсами вычислительной системы.

3) Microsoft Windows NT, базируясь на платформе Win32, обладает всеми ее преимуществами – многозадачность/многопоточность, безопасная работа с ресурсами вычислительной системы.

4) Пакет *Microsoft Visual Studio 98*, выбранный в качестве инструментальной среды разработки, работает на платформе Win32.

вернуться на **“Содержание”**

3.4 Интерфейс связи с МПСУ

3..4.1 В силу того, что «Протокол обмена МПСУ» в настоящее время является единственным имеющимся и рекомендуемым интерфейсом связи МПСУ с «верхним» уровнем, на котором запускаются прикладные программы, для OPC-сервера МПСУ использован именно он. «Протокол обмена МПСУ» обеспечивает соответствующий внутренний стандарт, который поддерживается супервизором МПСУ («супервизор» это программа, расположенная в контроллере МПСУ (модуле CPU) типа M231, M251…M255). Супервизор обеспечивает конфигурирование и стартовую диагностику МПСУ, взаимодействие с «верхним» уровнем по последовательным и параллельным каналам связи, выполнение функций управления УСО, принятых от ЭВМ верхнего уровня.

3.4.2 «Протокол обмена МПСУ» предоставляет возможность организации полного взаимодействия с МПСУ, а именно:

1) посылка запросов на выполнение супервизором операций МПСУ (служебных операций, операций модулей УСО, тестовых операций модулей УСО)

2) прием результатов отправленных запросов на выполнение операций (результаты выполнения служебных операций МПСУ, результаты выполнения операций модулей УСО, результаты тестирования модулей УСО)
3) контроль над обменом данными с МПСУ, с возможностью выявления ошибок и причин их возникновения.

Возможность работы с протоколом обеспечивает функция *sendbyt,* прототип которой приведен в **Приложении Б «Прототип функции обмена данными с МПСУ»**.

3.4.3 Протокол использует технологию обмена между главным устройством (master) и ведомым (slave), в рамках которой только одно устройство (master) может инициализировать цикл запрос / ответ. Подчиненные устройства (МПСУ) выполняют действия, указанные в запросе, и передают ответ, поставляя запрошенные данные источнику запроса. Следует отметить, что «Протокол обмена МПСУ» предполагает подключение одного МПСУ к последовательному или параллельному каналу связи компьютера, в нём нет указателя на номер или код устройства, как это имеется, например, в протоколе Modbus для последовательных каналов. Это означает, что при необходимости управлять несколькими МПСУ они включаются по схеме «звезда», а выбор МПСУ осуществляется путём выбора соответствующего канала. Приложение может работать как на компьютере, в котором установлен OPC-сервер и к которому подключены МПСУ, так и на другом компьютере (компьютерах) в локальной сети.

Коротко рассмотрим основные моменты «Протокола обмена МПСУ».

Формат запроса, используемый в реализации OPC-сервера МПСУ, обеспечивает передачу в МПСУ одного слова данных, то есть применяется второй формат запроса, который не предполагает передачу массива данных и, соответственно, не передаёт реквизит (фрагмент) «объём данных»:

Таблица 1. *Формат запроса*

Номер фрагмента	Число байтов	Назначение фрагмента
1	1	Индекс модуля
2	1	Номер модуля УСО данного типа
3	1	Код операции
4	1	Параметр
5	2	Блок данных – 2 байта (слово)

Индекс модуля – десятичное число, однозначно определяющее тип модуля.

Номер модуля УСО данного типа – логический номер модуля, который предназначен для идентификации модуля в каркасе, если модулей такого типа установлено несколько штук.

Код операции – ASCII символ, определяющий операцию, выполняемую в МПСУ после получения запроса.

Параметр – номер канала, который задает конкретный канал в модуле УСО, имеющем более одного канала.

Блок данных – данные (слово), которые сопровождают операцию.

После выполнения действий, заданных в запросе и подготовки возвращаемой информации МПСУ передает ответ. Формат ответа представлен в Таблице 2

Таблица 2 *Формат ответа*

Номер фрагмента	Число байт	Назначение фрагмента
1	2	Слово состояния процесса
2	2	Объем данных в байтах
3	N	Блок данных

Слово состояния процесса – определяет заключительное состояние выполнения операции в МПСУ, формируется супервизором МПСУ.

Значение слова состояния 0001hex говорит о том, что процесс выполнения команды завершен нормально, значение 0002hex говорит о том, что при выполнении команды зафиксирована ошибка.

Объем данных – объем данных ответа МПСУ в байтах.

Блок данных – N байт данных, сформированных МПСУ во время выполнения команды.

3.4.4 Прикладной уровень «Протокола обмена МПСУ» реализован в виде функции SENDBYT, которая использована при написании функций библиотеки *dataserv.dll*.

Дисциплина передачи запроса из PC и приема ответа из МПСУ следующая:

- Первый вызов функции производит инициализацию обмена (открытие порта)
- Второй вызов функции производит контроль обмена
- Третий вызов функции производит деинициализацию обмена (закрытие порта).

Значение, возвращаемое функцией, информирует о статусе выполнения той или иной стадии обмена.

вернуться на **“Содержание”**

3.5 Структура ОРС-сервера и назначение составных частей

3.5.1 Структура ОРС-сервера определяется функциями, которые он должен выполнять, внешними информационными потоками между ОРС клиентами и ОРС-сервером, ОРС сервером и МПСУ. В результате разработки получена структурная схема, изображенная на Рис.3 настоящего руководства.

3.5.2 Верхний уровень системы предназначен для предоставления возможности обмена данными ОРС-сервера с ОРС клиентами. Данный уровень представляет собой COM объекты и COM интерфейсы, регламентированные спецификациями OPC Data Access Custom Interface Standard и OPC Data Access Automation Interface Standard. По сути, верхний уровень предоставляет доступ к пространству тегов ОРС-сервера, то есть к адресам данных ОРС-сервера

3.5.3 В основе МПСУ заложен магистрально-модульный принцип, позволяющий легко осуществить конфигурирование системы под конкретную задачу, выбирая состав аппаратных модулей из обширной существующей номенклатуры. Пространство тегов ОРС сервера должно отражать конфигурацию МПСУ, т.е. состав модулей, входящих в МПСУ, а так же структуру самих модулей УСО (каналы ввода/вывода). Более того, конфигурирование пространства тегов ОРС сервера должно происходить автоматически, без вмешательства пользователя. Для решения этой задачи в структуре выделены ***Модуль конфигурирования ОРС сервера,*** и так называемый ***Виртуальный образ МПСУ***.

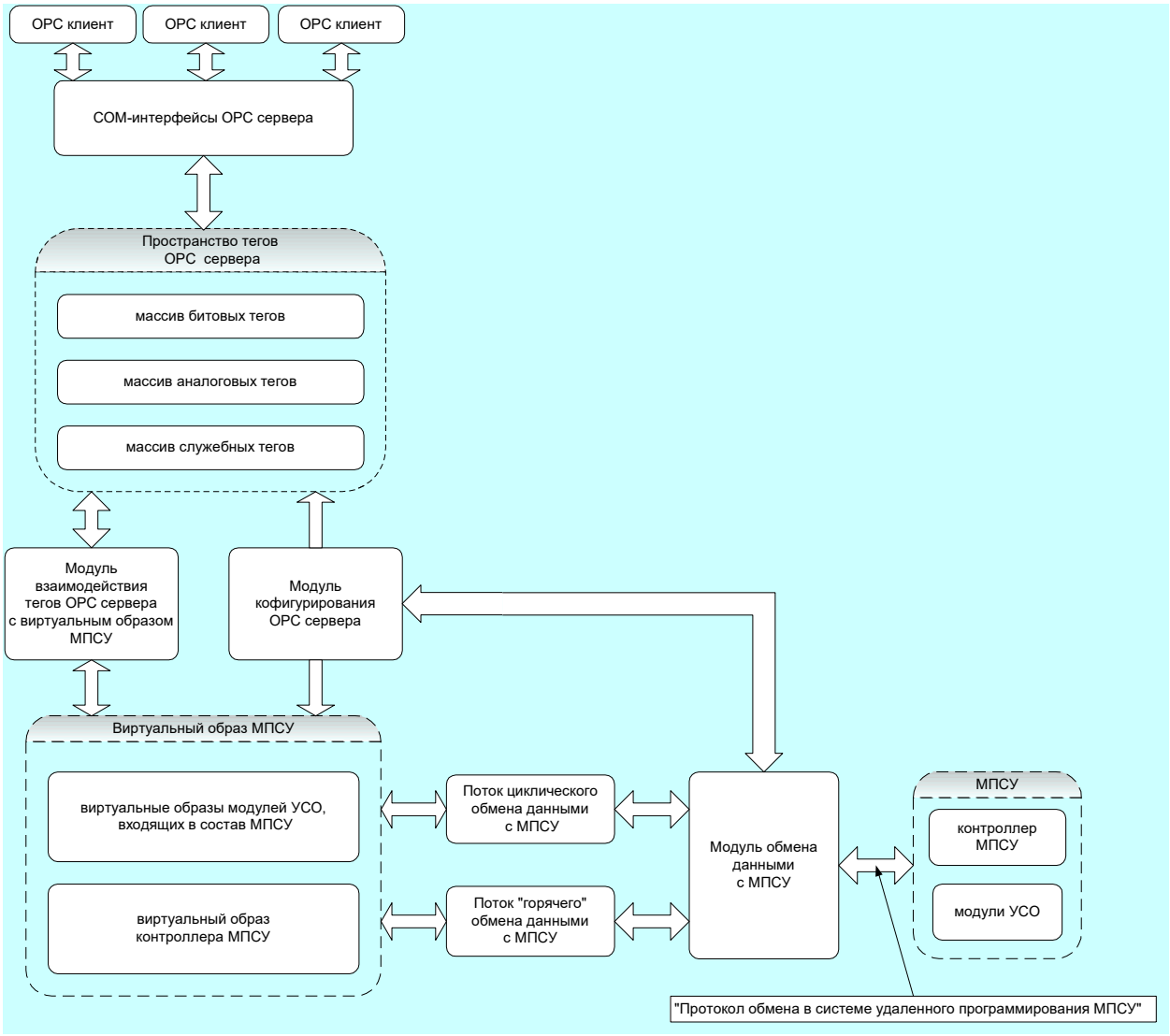

Рис.3 Структурная схема ОРС сервера.

Функции ***Модуля конфигурирования ОРС сервера*** получает информацию от МПСУ о составе входящих (установленных) модулей, создаёт соответствующий полученной информации виртуальный образ МПСУ и настраивает имена тегов ОРС-сервера в строгом соответствии с созданным виртуальным образом МПСУ, отражая в именах тегов конфигурацию МПСУ.

Виртуальный образ МПСУ хранит информацию о составе модулей МПСУ, полученную в результате последнего запроса конфигурации МПСУ. Кроме того, ***Виртуальный образ МПСУ*** инкапсулирует операции, выполняемые модулями УСО – аналоговый ввод / вывод, дискретный ввод / вывод (другие операции пока не включены). Это означает, что *виртуальный образ МПСУ* формирует запросы к МПСУ на выполнение операций модулей УСО, кроме того, он расшифровывает результат выполнения этих запросов (ответы МПСУ).

3.5.4 Для выполнения обмена между массивом тегов ОРС-сервера и *виртуальным образом МПСУ* в структуру системы введен ***Модуль взаимодействия тегов ОРС-сервера с Виртуальным образом МПСУ***. Задача этого модуля состоит в делегировании запросов на изменение значения тегов *виртуальному образу МПСУ*, а также в обеспечении обратного

информационного потока – преобразование данных *виртуального образа МПСУ* в значения и свойства тегов ОРС-сервера.

3.5.5 Поскольку система обслуживает модули ввода и вывода (аналогового и дискретного) в ОРС-сервере предусмотрены два потока обмена данными с МПСУ: ***поток циклического обмена данными с МПСУ*** (постоянный опрос каналов модулей УСО аналогового и дискретного ввода, входящих в состав МПСУ) и ***поток «горячего» обмена данными*** (запись в каналы модулей УСО аналогового и дискретного вывода). ***Поток «горячего» обмена данными*** используется также для выполнения запросов на выполнение служебных операций модулей МПСУ (запрос на получение информации о конфигурации МПСУ, тестирование модулей УСО и так далее). Два указанных потока обмена данными выполняются параллельно во времени и использует один физический канал для обмена с конкретным МПСУ. По этой причине предусмотрена синхронизация этих потоков. Для выполнения задачи обмена данными потоков с МПСУ посредством «Протокола обмена МПСУ» (Протокола обмена в системе удаленного программирования МПСУ), а также задачи синхронизации потоков в структуру системы введен ***Модуль обмена данными с МПСУ***.

вернуться на **“Содержание”**

3.6 Укрупненный алгоритм функционирования ОРС-сервера

Укрупненный алгоритм функционирования системы представлен блок-схемой на Рис.4 настоящего руководства.

1) Первоначально выполняется запрос конфигурации МПСУ, затем на основе полученной информации создается *виртуальный образ МПСУ*. На данном этапе формируется поток циклического обмена с МПСУ: в поток включаются образы модулей УСО, выполняющих операции аналогового и дискретного ввода. После создания *виртуального образа МПСУ* выполняется также автоматическая настройка пространства тегов ОРС-сервера: пространство тегов приводится в строгое соответствие с созданным *виртуальным образом МПСУ* (достигается это путем создания групп тегов ОРС-сервера, а так же и соответствующих именований тегов). После выполнения описанных действий производится запуск потока циклического обмена с МПСУ, потока «горячего» обмена с МПСУ, выполняется запуск механизмов синхронизации этих потоков.

2) Функционирование после запуска потоков обмена с МПСУ

При отсутствии запросов со стороны ОРС клиентов на запись новых данных в каналы модулей аналогового или дискретного вывода выполнятся циклический обмен данными с МПСУ. При этом *виртуальный образ МПСУ* получает данные о состоянии каналов модулей аналогового и дискретного ввода, преобразует эти данные в надлежащую форму и модуль взаимодействия пространства тегов с *виртуальным образом МПСУ* изменяет соответствующие теги ОРС-сервера.

Как только со стороны ОРС клиента возникает запрос на запись в канал (каналы) модуля (модулей) аналогового или дискретного вывода, производится делегирование этого запроса *виртуальному образу МПСУ*. *Виртуальный образ МПСУ* готовит запрос на выполнение операций по записи нового значения данных в канал модуля УСО и помещает этот запрос в поток «горячего» обмена данным с МПСУ. Поток «горячего» обмена дожидается освобождения ресурсов физического канала обмена данными с МПСУ и перехватывает его. По завершении выполнения обмена данным с МПСУ поток «горячего» обмена освобождает ресурсы физического канала обмена данными с МПСУ.

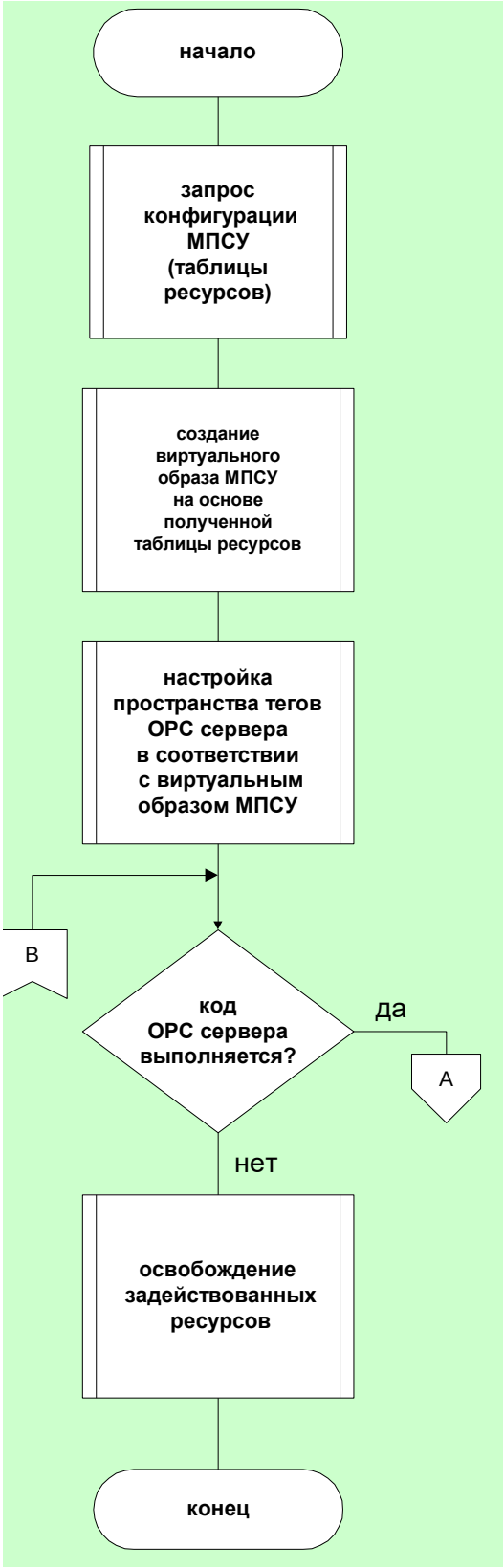
начало
запрос конфигурации МПСУ (таблицы ресурсов)
создание виртуального образа МПСУ на основе полученной таблицы ресурсов
настройка пространства тегов ОРС сервера в соответствии с виртуальным образом МПСУ
В
код ОРС сервера выполняется?
да
А
нет
освобождение задействованных ресурсов
конец

Рис.4

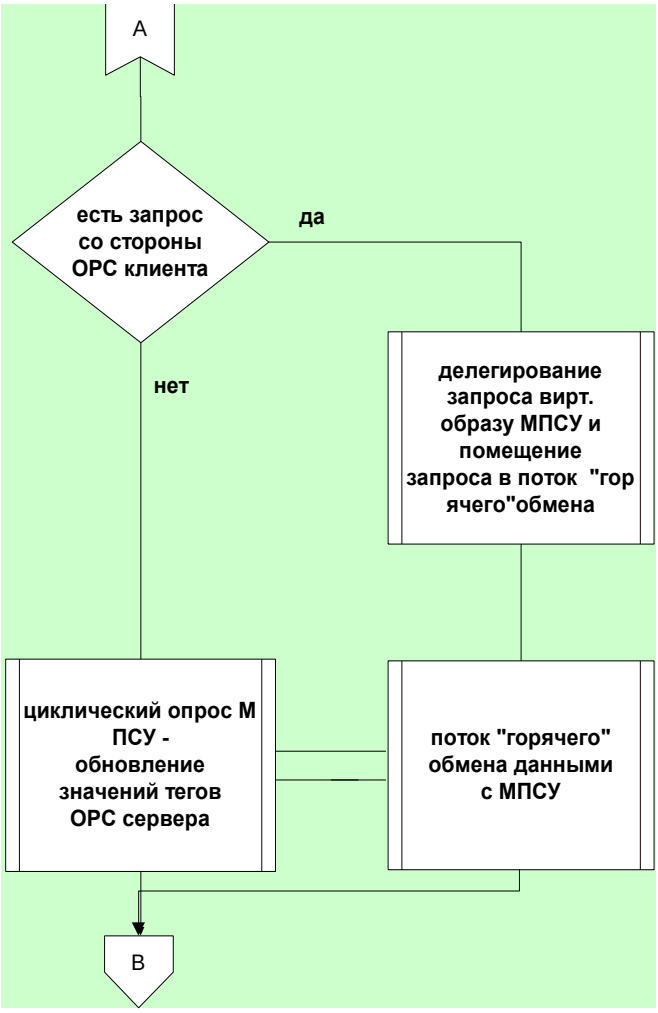

Рис.4 (продолжение)

вернуться на **“Содержание”**

3.7 Реализация модулей ОРС-сервера

Большинство модулей ОРС-сервера в виде С++ классов, что обусловлено применением ООП (объектно-ориентированного подхода) при разработке.

1) Уровень взаимодействия ОРС клиентов с ОРС-сервером.

Верхний уровень проектируемой системы, а именно COM объекты и COM интерфейсы реализованы в виде исполняемого файла **UniOPC.exe** (данный файл входит в комплект поставки **Fastwell UniOPC)**. COM объекты и COM интерфейсы, реализованные в UniOPC.exe, предоставляют ОРС клиентам доступ к пространству тегов в соответствии со спецификациями OPC Data Access Custom Interface Standard и OPC Data Access Automation Interface Standard.

2) Пространство тегов ОРС сервера

Пространство тегов (целочисленных, вещественных и битовых) ОРС-сервера представлено в виде массивов экземпляров структур **TIntTag**, **TAnTag** и **TBitTag** соответственно. Структуры тегов определены в листинге файла ***dataserv.h*** (см. Приложение А настоящего руководства).

3) Модуль взаимодействия пространства тегов ОРС-сервера с виртуальным образом МПСУ

Данный модуль представлен в виде трех функций:

SetAnTag

SetIntTag

SetBitTag

Эти функции экспортируются библиотекой dataserv.dll. Прототипы функций, а так же их определение представлены в Приложении А настоящего руководства. Функции осуществляют делегирование запросов ОРС клиентов к *виртуальному образу МПСУ* на запись в каналы модулей МПСУ. Блок-схема алгоритма функционирования этого модуля представлена на Рис.4 настоящего руководства.

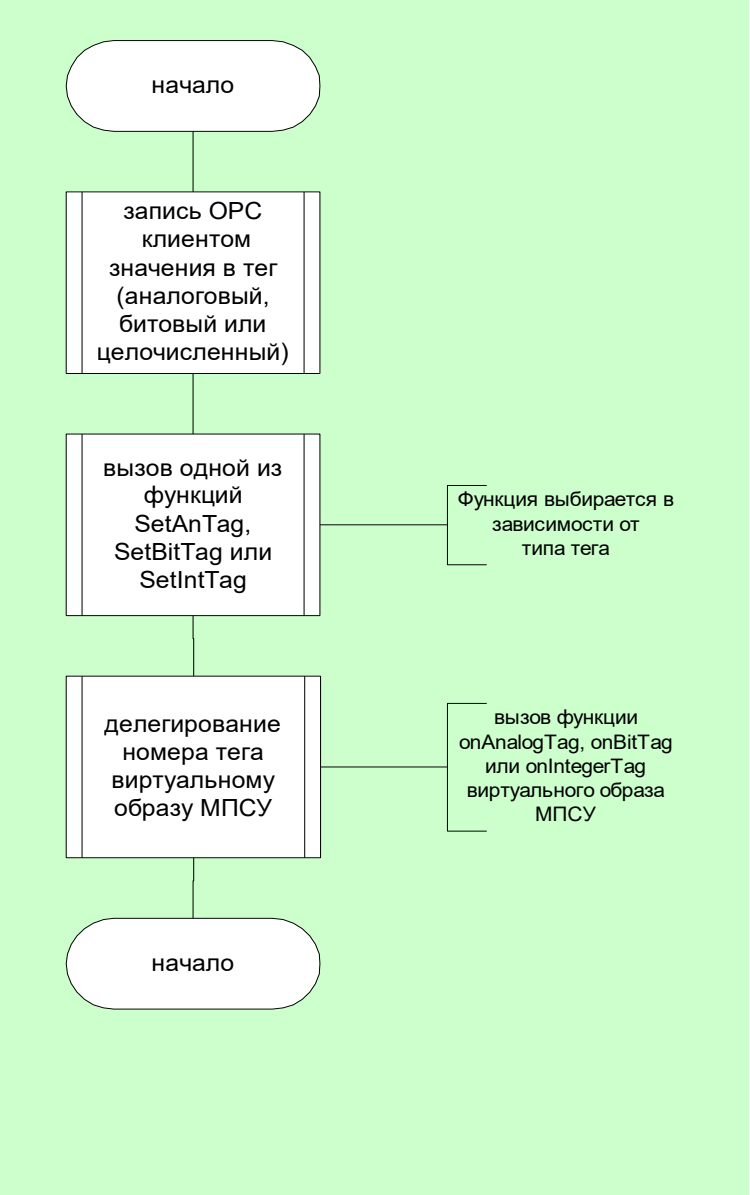

Рис.5 Алгоритм функционирования модуля взаимодействия пространства тегов ОРС сервера с виртуальным образом МПСУ

При записи ОРС клиентом значения в тег ОРС-сервера происходит вызов одной из функций, экспортируемых библиотекой dataserv.dll. Конкретное имя функции зависит от типа тега, значение которого изменено. Соответственно: SetAnTag для аналогового тега, , SetIntTag для целочисленного тега, SetBitTag для битового тега. Код этих функций делегирует номер тега, значение которого изменилось (идентификатор тега в пространстве тегов ОРС сервера) *виртуальному образу МПСУ*.

4) Виртуальный образ МПСУ и модуль конфигурирования ОРС сервера.

Данные модули являются наиболее сложными по набору выполняемых функций. *Виртуальный образ МПСУ*, а также модуль конфигурирования ОРС сервера представлен в виде С++ класса Csystem (определение класса приведено в Приложении В настоящего руководства). *Виртуальный образ МПСУ* представляет собой набор виртуальных образов модулей УСО. Виртуальные образы МПСУ представлены в виде С++ классов.

Базовым для классов модулей МПСУ является абстрактный класс Cdevice (определение класса приведено в Приложении В настоящего руководства). Он содержит объявление членов данных, а также методов, являющихся общими для всех классов модулей УСО. Наиболее важными членами этого класса являются:

CDevice::m_pSystem – указатель на объект *виртуального образа МПСУ*, к которому принадлежит данный модуль

CDevice::m_tagTimeOut – указатель на тег, содержащий значение таймаута выполнения операций модуля УСО.

CDevice::m_tagValid – указатель на тег, содержащий статус модуля УСО (получаем в результате выполнения запроса операции тестирования модуля УСО)

Наиболее важными методами этого класса являются следующие:

CDevice::CDevice – конструктор, выполняющий инициализацию членов класса.

CDevice:: preInitModule – метод, выполняющий действия, предшествующие инициализации объекта класса модуля УСО.

CDevice:: IntitiateModule – виртуальный метод, подлежащий перегрузке в классе модуля УСО. Назначение этого метода: окончательная инициализация объекта класса модуля УСО, то есть создание группы тегов ОРС сервера, отождествляемой с данным модулем УСО (объектом класса модуля УСО), запись адресов тегов в памяти процесса в соответствующие члены класса (указатели на теги модуля УСО). А также стартовая установка значений тегов (значение таймаута, результат стартового тестирования модуля УСО).

CDevice::ExecuteCommand – виртуальный метод, подлежащий перегрузке в классе модуля УСО. Назначение этого метода: подготовка запросов на выполнение операций модуля УСО и извлечение информации из данных, полученных по завершению выполнения запроса, преобразование извлеченной информации в значения тегов.

CDevice::TestModule - виртуальный метод, подлежащий перегрузке в классе модуля УСО. Назначение этого метода: выполнение запросов операции тестирования модуля УСО.

Классы модуля УСО являются образами конкретных модулей УСО (объявления и определения классов см. в Приложении В настоящего руководства):

CM113 - класс, соответствующий модулю аналогового ввода M113
CM101 – класс, соответствующий модулю дискретного ввода M101
CM102 – класс, соответствующий модулю дискретного вывода M102
CM103 - класс, соответствующий модулю релейного коммутатора (вывода) M103
CM201 - класс, соответствующий модулю дискретного ввода M201
CM202 - класс, соответствующий модулю дискретного вывода M202

CM203 - класс, соответствующий модулю дискретного вывода M203
CM204 – класс, соответствующий модулю маскированного аналогового ввода M204
CM210 - класс, соответствующий модулю аналогового вывода M210
CMY01 - класс, соответствующий дополнительному модулю пользователя MY01

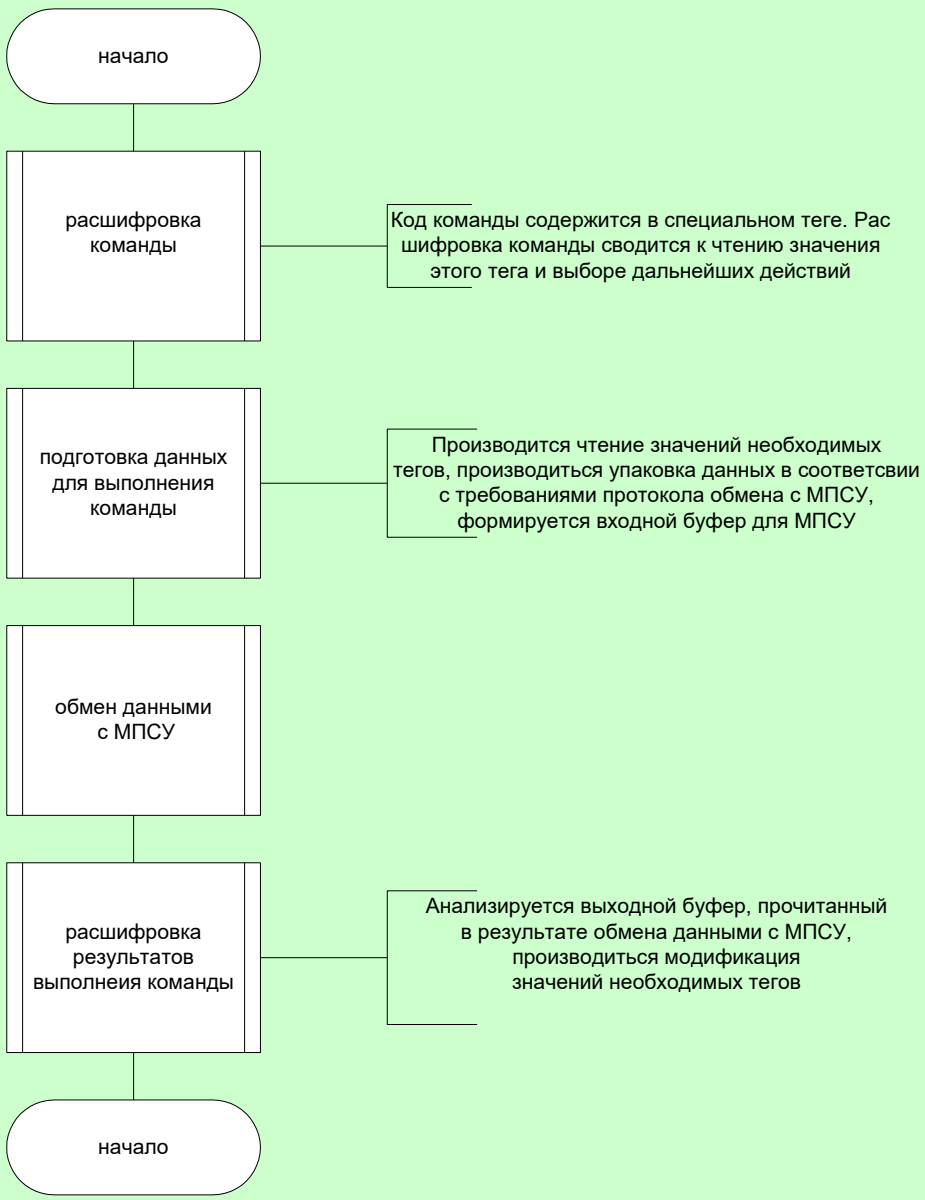

Рис. 6

Рассмотрим более подробно метод CDevice::ExecuteCommand. Код этого метода осуществляет операции, присущие модулям УСО. Конкретная реализация этого метода зависит от класса модуля УСО (конечного класса, типа модуля). Обобщенный алгоритм этого метода представлен на Рис.6 настоящего руководства. На первом этапе функционирования метода производится расшифровка команды, которую требуется выполнить. После того, как команда ясна, производится подготовка данных для выполнения запроса к МПСУ. Значения тегов, в которых содержится информация, необходимая для

выполнения операции конкретного модуля УСО, упаковываются в соответствии с форматом запроса к МПСУ. После подготовки всех необходимых данных осуществляется обмен данными с МПСУ (в соответствии с дисциплиной обмена РС и МПСУ по «Протоколу обмена МПСУ»). По завершении обмена производится обработка данных, полученных в результате выполнения команды в МПСУ. При этом значения соответствующих тегов устанавливаются в соответствии с результатом выполнения операции модулем УСО. Код методов для всех классов модулей УСО представлен в Приложении В настоящего руководства.

Рассмотрим более подробно класс CSystem. Метод класса CSystem::mcsIntiate (см. Приложение В настоящего руководства) выполняет запрос на получение так называемой таблицы ресурсов МПСУ, то есть информации о составе модулей УСО, установленныз в МПСУ. В случае успешного выполнения запроса на получение таблицы ресурсов управление передается методу класса CSystem::mcsRegister. Код данного метода извлекает информацию о конфигурации МПСУ, создавая объекты классов модулей УСО, соответствующих модулям УСО, входящим в состав МПСУ. При этом указатели на объекты классов, соответствующих модулям дискретного и аналогового ввода, помещаются в поток циклического обмена данными с МПСУ. После создания очередного объекта класса модуля УСО код метода производит настройку созданного объекта. В процессе настройки производится:

- создание группы в пространстве тегов OPC сервера, присвоение группе имени, соответствующего названию модуля и его логическому номеру в каркасе
- выделение из пространства тегов OPC сервера нужного количества не зарезервированных тегов, присвоение им имен, соответствующих названию каналов модуля УСО
- добавление тегов в группу
- сохранение указателей на выделенные теги в членах объекта класса модуля УСО, а так же сохранение идентификаторов тегов в объекте класса модуля УСО.

Блок-схема метода mcsRegister приведена на Рис.7 настоящего руководства.

Методы класса CSystem::onAnalog, CSystem::onBit, CSystem::onInteger (см Приложение В настоящего руководства) представляют своего рода интерфейс для взаимодействия с пространством тегов OPC-сервера. Именно этим методам делегируются запросы на изменение значений тегов OPC-сервера OPC клиентами. Код этих методов отыскивает объект класса модуля УСО, соответствующий идентификатору тега, и помещает указатель найденного объекта в поток «горячего» обмена данными с МПСУ.

5) Потоки обмена данными с МПСУ и модуль обмена данными с МПСУ

Потоки обмена данными с МПСУ реализованы на основе очереди типа FIFO. Для реализации очереди вводится параметризированный класс динамического двунаправленного линейного связного списка Clist (см. Приложение Г настоящег руководства).

В список потока циклического обмена данными с МПСУ помещаются указатели на объекты классов модулей УСО, осуществляющих дискретный и аналоговый ввод. Инициализация этого списка производится при конфигурировании OPC-сервера – метод mcsRegister класса CSystem.

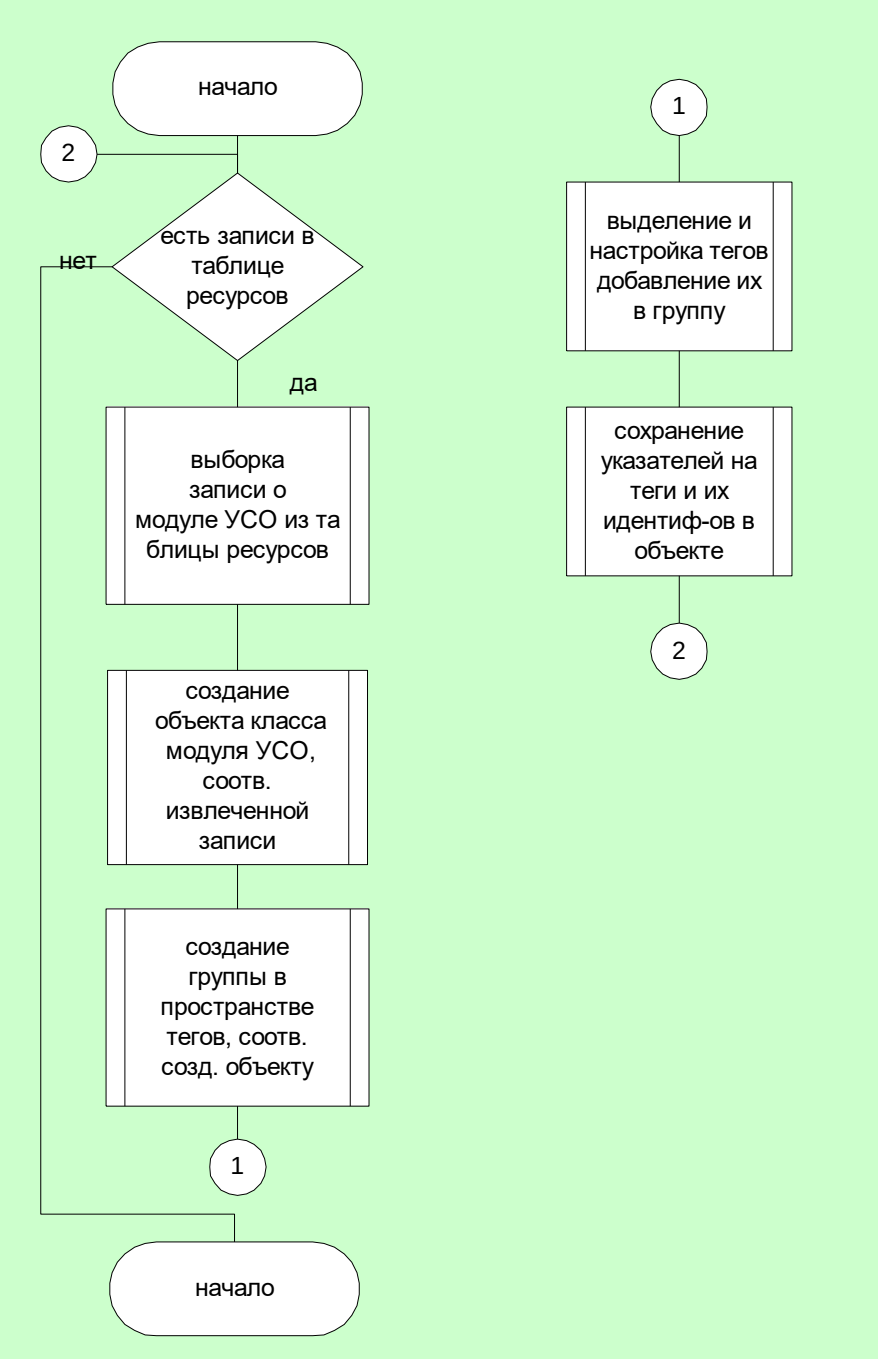

Рис.7 Блок-схема алгоритма метода CSystem::mcsRegister

В список «горячего» обмена данными с МПСУ указатели на объекты классов модулей помещаются при возникновении необходимости в этом, а именно при записи значения в теги ОРС-сервера ОРС клиентами новых значений. При этом следует отметить, что, помимо указателя на объект класса модуля УСО в списке сохраняются значения тегов, соответствующих данному объекту. Это необходимо делать, поскольку может сложиться ситуация, когда следующая запись значения в тег ОРС-сервера произойдет до выполнения предыдущего запроса на вывод данных. Для хранения такой информации введена структура TdeviceStruct (см. Приложение В настоящего руководства).

Сами потоки обмена ОРС-сервера с МПСУ реализованы с помощью двух потоков операционной системы. Соответственно, для потока циклического обмена потоковая функция CycleThread и для потока «горячего» обмена – QueryThread (см. Приложение Г настоящего руководства).

Синхронизация потоков выполнена с применением критической секции, указатель на объект которой храниться в классе виртуального образа МПСУ-CSystem::m_pReadWriteCritical (см. Приложение В настоящего руководства). Основной смысл синхронизации заключается в том, чтобы не допустить выполнение одного из потоков в то время, когда выполняется второй поток обмена данными с МПСУ. Потоки имеют одинаковый приоритет, поэтому полное вытеснение одного потока другим на длительный период времени невозможно.

Особенностью потока «горячего» обмена данными является то, что после обслуживания очередного элемента очереди, предыдущий элемент из неё удаляется.

В потоках производится выборка очередного указателя на объект класса модуля УСО, вход в критическую секцию, вызывается метод объекта ExecuteCommand. По возвращении управления обратно в функцию потока осуществляется выход из критической секции.

В классе CSystem предусмотрено управление потоками: останов потока и запуск потока. Выполняется это кодом методов CSystem::m_sysStopThreads и CSystem::m_sysStartThreads(см. Приложение 3).

вернуться на **“Содержание”**

3.6 Обмен данными ОРС севера с МПСУ

Как уже говорилось, обмен данными ОРС-сервера с МПСУ осуществляется посредством «Протокола обмена МПСУ». Модуль обмена данными с МПСУ реализован в методе класса CSystem CSystem::SendCommand (см. Приложение В настоящего руководства).

Код данного метода осуществляет обмен в соответствии с описанием прикладного уровня «Протокола обмена МПСУ». Блок-схема алгоритма метода приведена на Рис.8 настоящего руководства.

Первым действием является открытие порта связи с МПСУ–COM1-COM4, LPT, PRT – при этом какой именно порт открывается определяется членом переменной класса CSystem CSystem::m_PortNumber.

В ОРС-сервере МПСУ предусмотрена ситуация, при которой к одному компьютеру, на котором функционирует ОРС-сервер, может быть подключено несколько контроллеров МПСУ. Это достигается созданием нескольких экземпляров класса CSystem. Каждый экземпляр соответствует тому или иному порту связи с МПСУ (до 4-х последовательных COM-портов и 2 параллельных порта). Идентификатор порта сохраняется в члене класса CSystem CSystem::m_PortNumber. Значение данного члена класса используется прежде всего при обмене данными с МПСУ. Кроме того, номер порта влияет на название корневой группы в пространстве тегов ОРС сервера, а также на названия тегов. Создание экземпляров класса CSystem производится в конструкторе класса CInterActScope::CinterActScope (см Приложение В настоящего руководства).

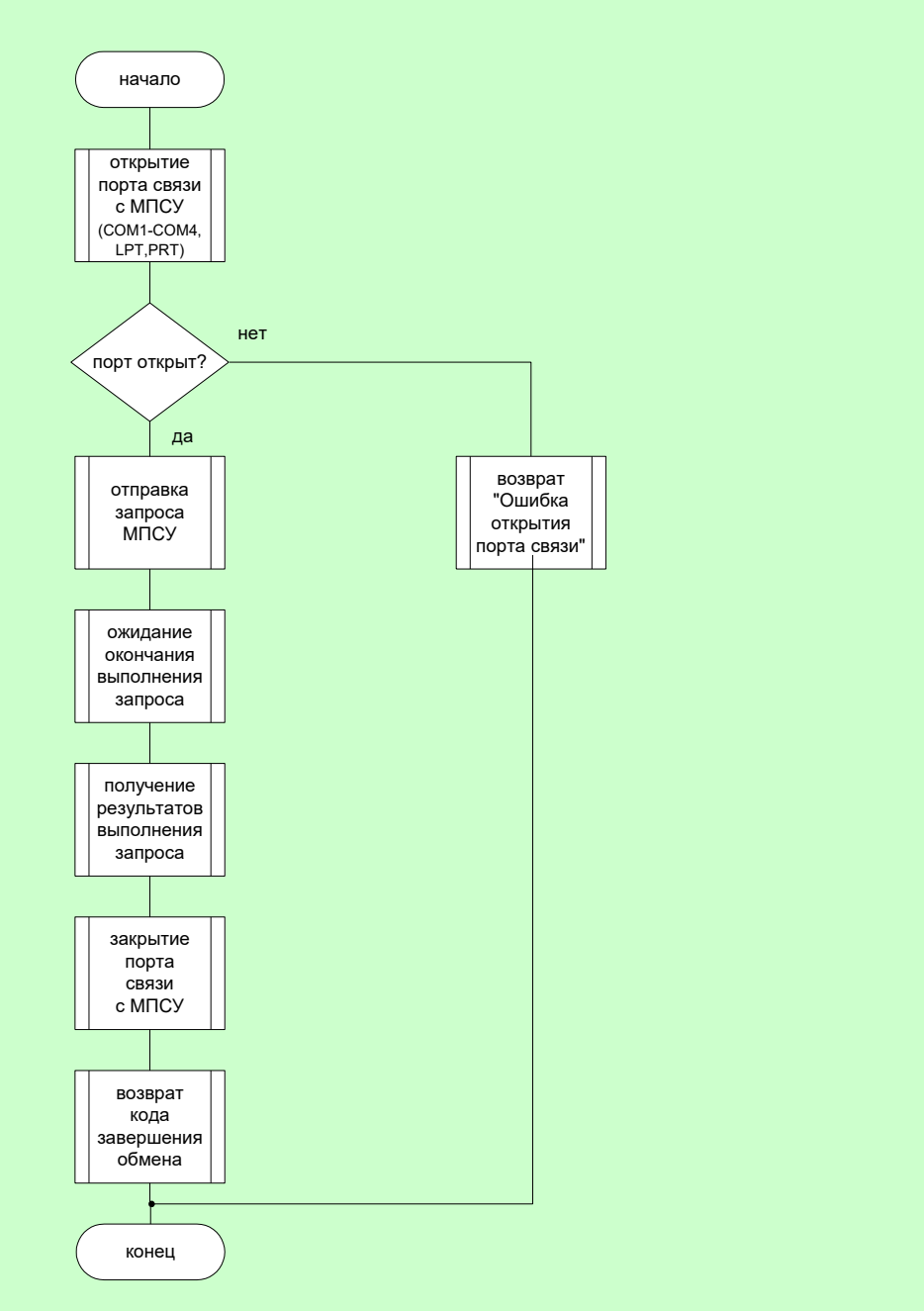

Рис.8 Блок-схема алгоритма метода CSystem::SendCommand

вернуться на **“Содержание”**

4 Средства управления и тестирования ОРС-сервера МПСУ

вернуться на **“Содержание”**

4.1 Средства управления ОРС-сервера

4.1.1 Цели разработки пользовательского программного интерфейса для управления ОРС-сервером МПСУ:

1) Предоставить максимально удобный доступ к функциям ОРС сервера, который обеспечивается наличием меню команд и панели инструментов, предоставляющих доступ к функциям ОРС сервера.

2) Избавить пользователя от необходимости вручную прописывать имена тегов, адреса источников данных тегов. Это достигается путем реализации функции автоматического создания тегов. Доступ к данной функции предоставляется через главное меню ОРС-сервера.

Поскольку программный продукт ориентирован на работу с аппаратурой, то пользователю предоставляется возможность получать визуальную информацию о состоянии аппаратуры (невозможно полностью исключить вероятности сбоев и неисправностей). Эта цель достигается путем введения специальных служебных тегов, значения которых пользователь может наблюдать в режиме «Монитор»

Программа имеет оконный интерфейс, главное окно программы представлено на Рис.9 настоящего руководства.

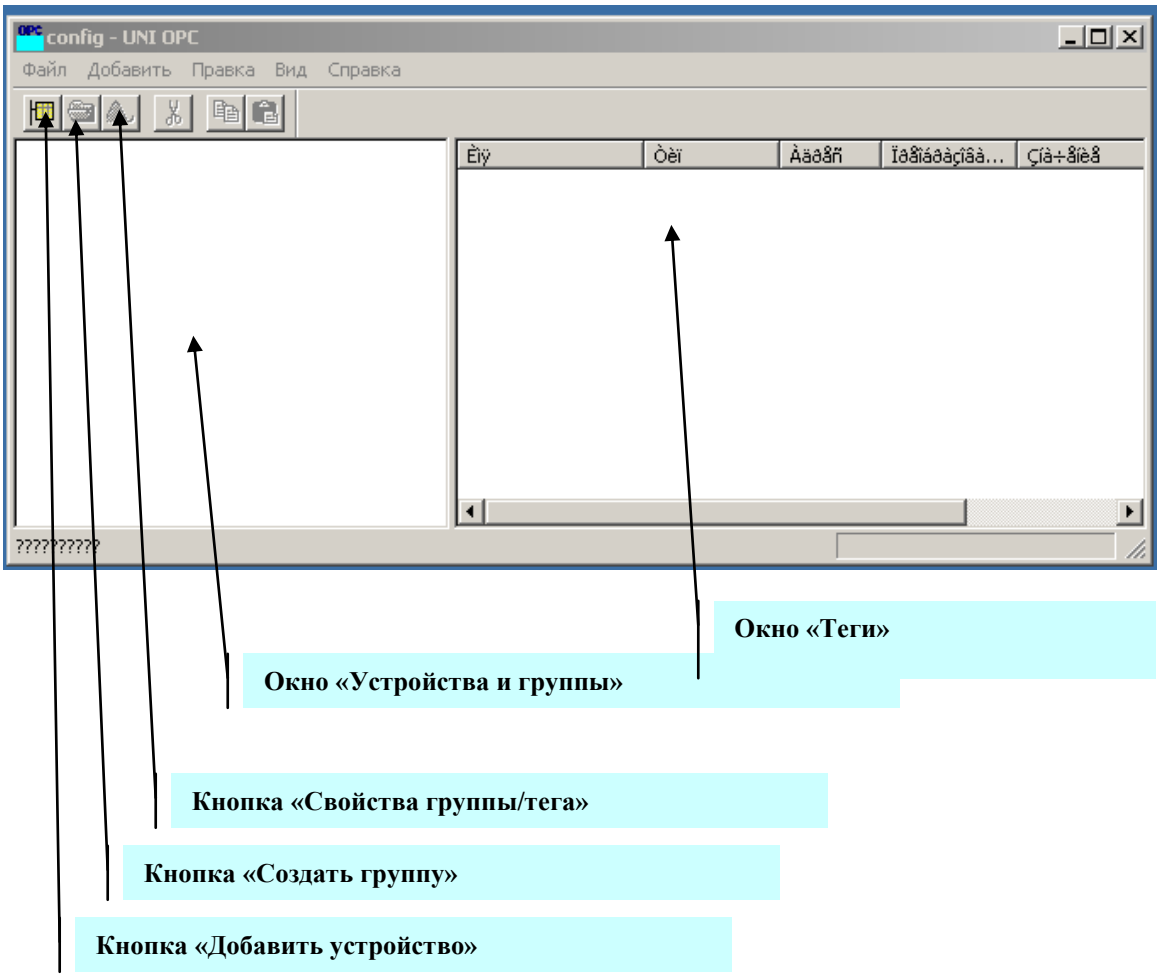

Рис.9 Главное окно ОРС сервера для МПСУ

4.1.2 Описание органов управления ОРС сервером.

Управление ОРС сервером производится с помощью меню и панели инструментов. Действия, производимые при выборе того или иного пункта меню и панели инструментов, приведены ниже:

- Меню **Файл**

◦ **Создать** – создание новой конфигурации ОРС-сервера
◦ **Открыть…** - открытие ранее сохраненной конфигурации ОРС-сервера
◦ **Сохранить** – сохранение текущей конфигурации ОРС-сервера с именем файла по умолчанию.
◦ **Сохранить как…** - сохранение текущей конфигурации ОРС-сервера с изменением имени файла.
◦ **Выход –**завершение работы по настройке ОРС-сервера МПСУ

- Меню **Добавить**

◦ **Устройство** (кнопка **«Добавить устройство»** панели инструментов) - выполнение запроса таблицы ресурсов МПСУ (составе модулей УСО, их состояние)
◦ **Группа (**кнопка **«Добавить группу»** панели инструментов) – создание и добавление пустой группы тегов.
◦ **Тег** (кнопка **«Добавить тег»** панели инструментов) **–** создание и добавление в текущую группу нового тега. При этом в появляющемся диалоговом окне пользователю предлагается:
-◦ ввести имя тега
-◦ выбрать тип тега (целочисленный 16-разрядный, целочисленный 32-разрядный, вещественный, битовый)
-◦ задать границы значений для тега, масштабирование значение тега
-◦ указать адрес в пространстве тегов ОРС сервера, по которому берется значение нового тега. Это удобно в том случае, когда необходимо иметь два тега с разными именами, но одинаковым источником значения (переменной).
◦ **Создать теги автоматически –** данный пункт меню позволяет выполнить автоматическую настройку пространства тегов ОРС сервера в соответствии с конфигурациями каркасов МПСУ, подключенных к компьютеру. При выборе этого пункта меню производится создание и добавление групп и тегов, при этом названия групп и тегов формируются таким образом, что отражают состав модулей УСО, уставленных в каркасы МПСУ.

- **Меню Правка**

◦ **Вырезать –** удаление выделенного объекта (группы, тега)
◦ **Копировать** – копирование выделенного объекта в буфер обмена
◦ **Вставить –** вставка объекта, скопированного ранее в буфер обмена
◦ **Свойства** – вызов диалога редактирования свойств выделенного объекта. Диалог редактирования свойств позволяет редактировать следующие свойства:

- Имя группы или тега
- Частоту обновления значений тегов в группе
- Тип тега (целочисленный 16-разр, целочисленный 32-разр, вещественный, битовый)
- Границы значений тега и масштабирование значений тега
- Адрес источника данных тега

- Меню **Вид**

◦ **Монитор –** режим отображения информации в окне тегов OPC-сервера. При активизации данного пункта меню в окне тегов выводится их текущее значение, частота обновления тегов зависит от соответствующих установок для группы.
◦ **Меню Справка –** доступ к справочной службе OPC-сервера.

Главное окно программы разбито на два окна – «Устройства и группы» и окно «Теги». В окне «Устройства и группы» графически представлена информация об имеющихся группах тегов OPC-сервера. Иерархия дерева групп отражает структуру каркасов МПСУ, подключенных к компьютеру.

В окне «Теги» отображается информация о тегах выбранной в данный момент группы. Информация о тегах имеет следующий состав:

- Имя тега
- Тип тега
- Полный адрес тега в пространстве тегов OPC-сервера
- Преобразование (границы и масштабирование значения тега)
- Значение (в режиме «Монитор»)
- Описание

При смене текущей группы состав тегов, отображаемых в окне тегов, изменяется в соответствии с составом выбранной группы.

4.1.3 Конфигурирование OPC-сервера

OPC-сервер МПСУ реализован как smartOPC сервер. Это означает, что конфигурирование пространства тегов происходит автоматически. Для того чтобы сконфигурировать пространство тегов OPC сервера, необходимо выполнить следующие действия:

1) Добавить устройство, выбрав соответствующий пункт в меню «Добавить» или нажав кнопку «Добавить устройство» на панели инструментов. При этом OPC-сервер запрашивает информацию о конфигурации всех каркасов МПСУ, подключенных к компьютеру.

2) Выбрать пункт «Создать теги автоматически» в меню «Добавить». При этом на основе информации, полученной от каркасов МПСУ, OPC-сервер создаст группы и теги, присвоив им нужные имена и установив значения тегов по умолчанию.

После выполнения указанных действий пространство тегов OPC сервера для МПСУ будет сконфигурировано.

Группы, соответствующие модулям УСО, объединяются в свою очередь в группы в зависимости от принадлежности к каркасу МПСУ, подключенному к тому или иному порту связи с МПСУ.

Соответствие имен этих групп портам связи с МПСУ:

Frame_on_COM1 – контроллер МПСУ, подключенный к порту COM1
Frame_on_COM2 - контроллер МПСУ, подключенный к порту COM2
Frame_on_COM3 - контроллер МПСУ, подключенный к порту COM3
Frame_on_COM4 - контроллер МПСУ, подключенный к порту COM4
Frame_on_LPT - контроллер МПСУ, подключенный к порту LPT
Frame_on_PPRT - контроллер МПСУ, подключенный к порту PPRT

4.1.4 Описание групп, соответствующих модулям УСО МПСУ.

При автоматическом конфигурировании OPC сервера имена группам и тегам присваиваются с целью отразить структуру каркасов МПСУ.

Соответствие групп ОРС сервера модулям УСО:

M101_XX модуль M101
M102_XX модуль M102
M103_XX модуль M103
M113_XX модуль M113
M201_XX модуль M201
M202_XX модуль M202
M203_XX модуль M203
M204_XX модуль M204
M210_XX модуль M210
M230_XX модуль M230

Где XX – номер модуля УСО данного типа в каркасе МПСУ.

Во всех вышеперечисленных группах есть теги, являющиеся служебными тегами. Имена тегов и их описание следующие:

ErrCode – целочисленный тег, содержит код результата обмена данными с модулем УСО МПСУ, соответственно группе. Возможные значения и расшифровка этих значений:

0 – процесс обмена завершен нормально
1 – ошибка получения данных от МПСУ
2 – нет последнего служебного байта SV от МПСУ
3 – таймаут при передаче запроса
4 – таймаут при приеме ответа
5 – обмен не завершен, продолжается
6 – фатальный сбой связи

IsInThread - битовый тег, значение которого указывает ОРС-серверу включать значение **On**) или не включать (значение **Off**) данный модуль в поток циклического обмена данными с МПСУ.

Valid – целочисленный тег, значение которого формируется после тестирования модуля УСО. Значение данного тега содержит код выполнения теста модуля УСО. Расшифровка возможных значений тега:

0 – модуль функционирует нормально (положительный результат тестирования модуля УСО)
1 – Дефект СОЗУ контроллера
2 – Дефект ОЗУ контроллера
3 – Дефект типа «не сравнение данных»
4 – Дефект контрольного суммирования
5- Дефект основных команд контроллера
6 – Дефект команд расширенной арифметики
7 – Дефект типа «зависание»
8 – Дефект в тесте запланированных прерываний
9 – Дефект в тесте незапланированных прерываний
10 – Дефект «неверный символ»
11 – Дефект – «отсутствие готовности»

TimeOut – целочисленный тег, значение которого содержит таймаут ожидания в миллисекундах завершения выполнения операции для данного модуля УСО. По умолчанию для всех групп значение выставляется при автоматическом конфигурировании ОРС сервера 3000 миллисекунд.

вернуться на **“Содержание”**

4.1.5 Описание тегов, входящих в группу управления МПСУ Frame_on_NNN.

Command –целочисленный тег, запись значения в который выполняет какие-либо действия над МПСУ или всеми группами соответствующих модулей УСО, входящих в каркас МПСУ.

Значение тега	Выполняемая операция
1	Рестарт МПСУ. Получение конфигурации МПСУ, сравнение с предыдущей конфигурацией (при отличиях производится модификация пространства тегов ОРС-сервера)
2	Останов потоков обмена данными с МПСУ.
3	Возобновление потоков обмена данным с МПСУ

Valid – целочисленный тег, содержащий статус МПСУ.
0 – МПСУ функционирует нормально
1 – в работе МПСУ замечены ошибки

Перечень тегов, специфических для всех модулей УСО, достаточно громоздок, поэтому приведен в Приложении Д настоящего руководства.

4.2 Отладка и тестирование ОРС-сервера

4.2.1 Тестирование функциональности ОРС-сервера МПСУ на ранних стадиях разработки производилось с помощью ОРС клиента, входящего в комплект поставки UniOPC для разработчика. ОРС клиент позволял производить запись и чтение значений тегов ОРС сервера. По завершении реализации ОРС-сервера МПСУ было произведено тестирование системы с помощью SCADA-системы TraceMode версии 5.0 (Инструментальный пакет и демо-версия монитора реального времени), которая бесплатно предоставляется фирмой AdAstra разработчикам. Данная SCADA-система имеет встроенную поддержку ОРС и может работать и в качестве ОРС клиента и в качестве ОРС сервера.

4.2.2 При тестировании из системы в TraceMode ОРС-сервера МПСУ в каркас модулей были установлены: модуль аналогового ввода М204 (1 модуль), модуль аналогового вывода М210 (1 модуль), а так же модуль дискретного ввода М201 и модуль дискретного вывода М203. Модуль вывода являлись источниками тестирующих сигналов, а модули ввода принимали эти сигналы. Соединение входов и выходов при помощи сигнальных кабелей обеспечило обмен аналоговыми сигналами по 8-ми дифференциальным каналам и дискретными сигналами по 32-м однобитным каналам. В проекте, разработанном в SCADA системе TraceMode, связь с вышеуказанным модулями осуществлялась посредством ОРС-сервера МПСУ. Стоит отметить, что при этом была задействована группа COM интерфейсов и COM объектов, регламентированная спецификацией *OPC Data Access Custom Interface Standard*.

В каналы модуля аналогового вывода осуществлялась запись значений, формируемая FBD программой, форма выходного сигнала имела при этом форму «пилы». Одновременно осуществлялось циклическое чтение состояния аналоговых входов модуля аналогового ввода М204. Состояние данных в каналах этих модулей было представлено на экране в виде графических трендов. Наблюдалось некоторое отставание изменения значений входных сигналов М204 от изменения значений выходов модуля М210. Форма сигналов отличалась несущественно. Задержки были обусловлены в первую очередь работой ядра TraceMode (в нём выполняется пересчет всей базы каналов), а также задержками на

передачу данных по последовательному каналу в МПСУ и обратно. Сбоев ОРС сервера или иных отклонений результатов от ожидаемых при тестировании не зафиксировано.

Для тестирования взаимодействия SCADA системы с дискретными модулями был разработан проект (FDB – программа), которая обеспечивает выполнение записи «бегущей единицы» в дискретные каналы модуля М203 и одновременное чтение дискретных входов модуля М201. Информация о состоянии каналов также представляется на экране в графическом виде. Отставание изменения состояния дискретных входов модуля М201 от изменения состояния дискретных выходов модуля М203 (что характеризует быстродействие всей системы) наблюдалось незначительное и обусловлено спецификой ядра системы TraceMode (пересчет всей базы каналов), а также задержкой в последовательном канале связи. Сбоев в работе ОРС сервера для МПСУ также не возникало.

В результате тестирования ОРС сервера с применением SCADA системы TraceMode версии 5.0 была проверена и доказана способность ОРС сервера для МПСУ предоставлять доступ к функциям модулей УСО посредством ОРС интерфейса, регламентированного спецификацией OPC Data Access Custom Interface Standard.

4.2.3 Для проверки способности ОРС сервера предоставлять доступ к функциям МПСУ посредством ОРС интерфейса, регламентированного спецификацией *OPC Data Access Automation Interface Standard* было разработано приложение на языке программирования *Visual Basic6.0*, поддерживающим OLE автоматизацию. При этом в тестировании использовался контроллер МПСУ той же конфигурации (те же модули ввода/вывода). В приложении формировалась последовательность аналоговых значений на выходах модуля М210 по закону синусоиды с периодом 0.5 с. и суммарной амплитудой дифференциальных выходов 10,23 В. Одновременно осуществлялось чтение аналоговых входов модуля М204. Информация о состоянии каналов аналоговых модулей была представлена в виде списка значений и графических трендов. Формы записываемого и читаемого сигналов мало отличались друг от друга. Задержки изменения сигналов на входах и выходах модулей были минимальны (фактически сказываются только задержки в последовательном канале связи).

Помимо циклической записи аналоговых значений в каналы модуля М210 была предусмотрена возможность записи значения пользователем. Изменения читаемого канала модуля М204 при этом следовали практически мгновенно за записью. Быстродействие ОРС-сервера МПСУ при работе, в частности, с аналоговыми модулями достаточно высоко, что является положительным аргументов в пользу его применения.

В тесте с применением дискретных модулей МПСУ также осуществлялась запись «бегущей единицы» в модуль дискретного вывода М203 при одновременном чтении состояния дискретных выходов модуля дискретного ввода М201. Задержек между изменениями значений не наблюдалось.

При тестировании ОРС сервера с помощью приложения ОРС клиента, разработанного на Visual Basic 6.0 была доказана способность ОРС-сервера предоставлять доступ к функциям модулей УСО посредством ОРС интерфейса, регламентированного спецификацией OPC Data Access Automation Interface Standard. Сбоев в работе ОРС сервера при тестировании с помощью приложения, разработанного на Visual Basic 6.0, или возникновения ситуации, отличной от ожидаемой, не возникало. Также было выявлено высокое быстродействие обмена данными с МПСУ посредством ОРС-сервера.

вернуться на **“Содержание”**

5 Заключение

В результате разработки OPC-сервера МПСУ удалось решить основную проблему интеграции компонентов промышленных систем управления: предоставление унифицированного интерфейса для взаимодействия отдельных составляющих системы. Созданный OPC-сервер предоставляет приложениям в операционной системе Windows NT стандартизованный интерфейс для доступа к функциям МПСУ. OPC-сервер МПСУ отвечает спецификациям OPC Data Access Custom Interface Standard и OPC Data Access Automation Interface Standard, что предоставляет возможность его применения в системах, построенных с применением современных SCADA и DSC систем.

Разработчикам и пользователям приложений верхнего уровня не нужно вникать в специфику обмена данными с МПСУ, поскольку специфика обмена данными с МПСУ скрыта внутри механизмов функционирования OPC-сервера МПСУ.

Разработка прикладных программ может выполняться на любых языках программирования, в которых реализована поддержка технологии COM или DCOM. При этом разработчику предоставляется возможность использования SCADA-системы с поддержкой OPC или два пути создания собственных приложений: с использованием интерфейсов автоматизации (например, на Visual Basic) или же с использованием традиционного программирования COM (Visual C++).

Поддержка DCOM (Distributed Component Object Model) в реализованном OPC сервере МПСУ делает возможным применение МПСУ в системах управления технологическими процессами с распределённой сетевой структурой компьютеров-станций (оператора, инженера, руководителя). Необходимость стыковки сетевых протоколов с МПСУ отпадает.

Высокое быстродействие OPC сервера для МПСУ позволяет использовать его в системах, где скорость обмена данными с аппаратурой является критичным фактором

Перспективы и пути совершенствования

1) В дальнейшем предполагается разработка OPC сервера для МПСУ на базе PC совместимого контроллера M260, M261;

2) При совершенствовании «Протокола обмена МПСУ» одновременно с внутренним ПО МПСУ, вероятно, будет необходимо вносить изменения в механизмы взаимодействия OPC сервера с МПСУ, другими словами, осуществлять поддержку данного программного продукта;

3) Пользуясь возможностью получения бесплатных демонстрационных версий различных SCADA-систем, новые образцы которых практически в обязательном порядке поддерживают обмен с OPC-сервером, представляется разумным выполнить тестовую проверку их взаимодействия с OPC-сервером МПСУ. Среди первых «кандидатов» систем можно назвать: InTouch (Factory Suite), КРУГ-2000/NT, GENIE, GENESIS 32, систему Citect фирмы Ci Technologies.

вернуться на “Содержание”

Приложение А. Каркас библиотеки dataserv.dll

Листинг файла dataserv.h

```
#ifndef DATASERV_H
#define DATASERV_H

#include <string.h>

#pragma pack(push, 1)
//---------- Наименование тега -----------//


struct TTagName
{
  char *name; // собственно наименование

  TTagName(){name=NULL;};
  ~TTagName() {delete name; name = NULL;};

  void setName( const char *aName )
    {
    delete name;
    if( !aName )
        { name = 0; return; };
    name = new char[ strlen( aName ) + 1 ];
    if(name)
      strcpy(name, aName );
    };
};

//------------------------ Аналоговый тег ------------------------//
struct TAnTag : TTagName
{
  float value; // значение аналогового тега
  TAnTag(){value=0;}
};

//------------------------ Битовый тег ------------------------//
struct TBitTag : TTagName
{
  unsigned char value; // значение тега (0 или 1)
  TBitTag(){value=0;}
};

//------------------------ Целочисленный тег ------------------------//

struct TIntTag : TTagName
{
  int value; // значение тега
  TIntTag(){value=0;};
};

//----------------- Данные для ОРС сервера --------------------//
struct TDataForOPCServer
{
  unsigned char dataIsValid;   // корректность данных
  unsigned short anTagsQnty;   // к-во аналоговых тегов
  const TAnTag *anTagsArray;   // ук-ль на массив аналоговых тегов
  unsigned short bitTagsQnty;  // к-во битовых тегов
  const TBitTag *bitTagsArray; // ук-ль на массив битовых тегов
  unsigned short intTagsQnty;  // к-во целочисленных тегов
  const TIntTag *intTagsArray; // ук-ль на массив целочисленных  //тегов

};
```

```
//=============== Функции, вызываемые OPC сервером ==============

extern "C"
{

//  1) функция, указывающая на расположение данных в памяти
__declspec( dllexport )
    const TDataForOPCServer * PASCAL GetDataForOPCServer( void );


//   2) функция для изменения значения аналогового тега
__declspec( dllexport )
      unsigned char PASCAL SetAnTagValue(
      unsigned  number //номер тега в пространстве тегов OPC сервера
      float value            //новое значение
 );

//   3) функция для изменения значения битового тега
__declspec( dllexport )
      unsigned char PASCAL SetBitTagValue(
      unsigned  number, //номер тега
      unsigned char value //новое значение
      );

//   4) функция для изменения значения целочисленного тега
__declspec( dllexport )
      unsigned char PASCAL SetIntTagValue(
      unsigned  number, //номер тега
      int value    //новое значение
      );

}; // end of extern "C"

#pragma pack(pop) // restore origiunal alignment

#endif
```

Листинг файла dataserv.cpp

```
__declspec( dllexport )
unsigned char PASCAL SetAnTagValue(
      unsigned  number,
      float value
   )
{
  if(number<0 || number>=MAX_TAG)
     return FALSE;
  AnTag[number].value=value;
  DWORD param=0;
  param = number;
   return (::scope->onAnalogTag(param)); //делегирование запроса OPC    //клиента виртуальному образу //МПСУ
};

//-----------------------------------------------------------------------------
__declspec( dllexport )
unsigned char PASCAL SetBitTagValue(
      unsigned  number,
      unsigned char value
      )
{
  if(number<0 || number>=MAX_TAG)
     return FALSE;
  BitTag[number].value=value;
  DWORD param=0;
  param=number;
  return (::scope->onBitTag( param)); //делегирование запроса OPC    //клиента виртуальному образу //МПСУ
```

```
};
//-----------------------------------------------------------------------------

__declspec( dllexport )
unsigned char PASCAL SetIntTagValue(
       unsigned  number,
       int value
       )
{
  if(number<0 || number>=MAX_TAG)
      return FALSE;
  IntTag[number].value=value;
  DWORD param = 0;
  param=number;
  return (::scope->onIntegerTag(param)); //делегирование запроса ОРС    //клиента виртуальному образу //МПСУ

};

//-----------------------------------------------------------------------------
```

вернуться на **“Содержание”**

Приложение Б. Прототип функции sendbyt

int sendbyt(int Mod_Open,
UINT NumberPort,
BYTE inputarray,*
UINT SzInput,
BYTE outputarray,*
UINT& MxSzOutput,
DWORD speed,
DWORD TimeOut)

Параметры функции:

NumberPort – номер порта для связи с МПСУ.
**inputarray* – указатель на буфер входных данных запроса
SzInput – размер буфера входных данных запроса
outputarray – указатель на буфер выходных данных запроса
SzOutput – ссылка на размер буфера выходных данных
speed – baudrate для порта связи с МПСУ
TimeOut- значение в миллисекундах таймаута выполнения запроса в МПСУ

вернуться на **“Содержание”**

Приложение В.
Исходный код классов CSystem, CDevice, классов модулей УСО

Объявление классов.

```
class CDevice
{
protected:
	void preInitModule(const char* prefix,void* param[]);
	void* m_pSystem; //pointer to CSystem object
	UINT SzInput;	// size of input data
	UINT SzOutput;	// size of output data
	BYTE* m_pInput;	//pointer to input array (command itself)
	BYTE* m_pOutput;	//pointer to output array (result of command execution)
	BYTE  m_Index;	//index of module
	BYTE  m_NumbFrm;	//number in frame
	int* m_tagTimeout;  //timeout for command execution
	int* m_tagErrorCode;//result of device test (state of device)
	int* m_tagValid;	//is this device valid (after restarting MCS) (request of device test)
	unsigned char* m_tagOnOff;//to control joining of device in read/write cycle(thread)
public:
	virtual void TestModule();
	CDevice(void* pSystem);
	CDevice(BYTE Index=0, BYTE NmbinFrm=0, void* mySystem = NULL);
	virtual ~CDevice();


	virtual bool IsInTheThread(); // tells about m_tagOnOff condition

	virtual bool ExecuteCommand(DWORD )=0; //execute commands featured to module
		//@@@ PARAMETERS ORDER IN ExecuteCommand METHOD @@@
		// param[0] - number of tag, which activates action(ignoring by modules objects)
		// param[1] - pointer to tag value(&SXxxTag::value)

	virtual void InitiateModule( void*[] )=0; // intialization of module
		//@@@ PARAMETERS ORDER IN void** @@@
		// param[0]  - m_tagTmout (timeout for operations of this module)
		// param[1]  - m_tagErrCode (result of last perfomed operation )
		// ...	- specific tags(information tags, controlling tags, etc.)
		// param[50] - m_tagOnOff (&BitTag[..]) - is this module joined in to Cycle Thread
		// param[51] - (void*)m_pSystem
		// param[52] - m_tagValid (pointer to IntTag[..]) is this module valid(after restarting MCS)
	friend class CSystem;
};


//////////////////////////////////////////////////////////////////
//base class for all analog modules
class CAnalog
{
	protected:
		UINT	Diapazon;
		double MantsBit;
	public:
		double DigitToAnalg(WORD Digital);
		WORD AnalgToDigit(double Analog);
		CAnalog(UINT D=0,double M=0) : Diapazon(D),MantsBit(M) {};
};
//////////////////////////////////////////////////////////////////////
//base class for all discrete modules
class CDiscrete
{
	protected:
		UINT WordLength;//word length of input/output
	public:
		CDiscrete( UINT wordLength=16){
			WordLength = wordLength;
```

```
                    };
                    ~CDiscrete() {};

                    WORD  WriteCode(unsigned char Discrete[]);  //returns WORD where all bits setted as need
                    void ReadCode( WORD Word, bool* temp);              //returns an array of bool type
};
//////////////////////////////////////////////////////////////////////
// All modules contatins pointers to values of tags
// This feature optimize work with tags
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
#define _M204_  6      //index of module in MCS
#define _M204_param 11 // vlm of void*[] array

//class simulate behaviour of M204 module
class CM204 : public CDevice, public CAnalog
{
protected:
          float* m_tagChanel[8]; //8 pointers to values of 8 analog tags
          int*        m_tagMask;             //pointer to mask for chanels to read
public:
          virtual bool ExecuteCommand(DWORD);
          CM204();
          CM204(BYTE index, BYTE numbinfrm, void** param);
          ~CM204();
          virtual void InitiateModule( void* pData[] );
private:
          inline void m204_GetData();
};

#define _M210_ 11      // index of module in frame
#define _M210_param 18 // vlm of void*[] array

class CM210 : public CDevice, public CAnalog
{
protected:
          float* m_tagChanel[16];
public:
          virtual bool ExecuteCommand(DWORD);
          CM210();
          CM210(BYTE index, BYTE numbinfrm,UINT TagNmb, void** param);
          virtual ~CM210();
          virtual void InitiateModule(void* pData[]);

private:
          UINT m_uiBaseTagNumber;
          int m210_GetChan(UINT);
};

//////////////////////////////////////////////////////////////
// macro definitions for common MCS commands
#define MCS_RESTART 1
#define MCS_THREADON 2
#define MCS_THREADOFF 3
#define MCS_INITAGAIN 4
//possible values of m_tagValid
#define MCS_CONFIGCHANGED 1
#define MCS_RESTARTFAILED 2
#define MCS_INITFAILED    3
#define MCS_OK            0

 //..for tests of devices

class CMCSFrame : public CDevice
{
private:
          int*  m_tagCommandChan; //int tag for commands to frame(see macro definitions some strings above)
```

```
public:
        virtual bool ExecuteCommand(DWORD);
        virtual void InitiateModule(void* param[]);
        CMCSFrame(void**);
        virtual ~CMCSFrame();
};

#define _M102_ 1
class CM102 :
        public CDevice,
        protected CDiscrete
{
public:
        virtual bool ExecuteCommand(DWORD);
        virtual void InitiateModule(void* param[]);
        CM102(BYTE index, BYTE number, void* param[]);
        virtual ~CM102();

protected:
        char prefix[50];
        unsigned char* m_tagChanel[16];
        unsigned char* m_tagWrteCode;
        virtual void SetPrefix(){ wsprintf(prefix, "M102_%2.2d.", m_NumbFrm); };
};

#define _M103_ 2
class CM103 : public CM102
{
protected:
        virtual void SetPrefix(){ wsprintf(prefix, "M103_%2.2d.", m_NumbFrm); };
public:
        CM103(BYTE index, BYTE number, void* param[]);
        virtual ~CM103();

};

#define _M201_ 3
class CM201 :
        public CDevice,
        protected CDiscrete
{
public:
        CM201(BYTE, BYTE, void*[]);
        virtual bool ExecuteCommand(DWORD);
        virtual void InitiateModule(void*param[]);
        CM201();
        virtual ~CM201();

protected:
        unsigned char* m_tagReadCode;
        unsigned char* m_tagChanel1[16];
        unsigned char* m_tagChanel0[16];
};


#define _M202_ 4
class CM202 :
        public CDevice,
        public CDiscrete
{
public:
        bool ExecuteCommand(DWORD);
        void InitiateModule(void* param[]);
        CM202(BYTE index, BYTE number, void* param[]);
        virtual ~CM202();

protected:
        unsigned char* m_tagWriteCode;
```

```
	unsigned char* m_tagChanel[8];
};


#define _M203_ 5
class CM203 :
	public CDevice,
	public CDiscrete
{
public:
	bool ExecuteCommand(DWORD);
	void InitiateModule(void* param[]);
	CM203(BYTE index, BYTE number, void* param[]);
	virtual ~CM203();

protected:
	int* m_tagChanN;
	unsigned char* m_tagWriteCode;
	unsigned char* m_tagChanel1[16];
	unsigned char* m_tagChanel0[16];
};

#define _M101_ 8
class CM101 :
public CDevice,
public CDiscrete
{
public:
	virtual bool ExecuteCommand(DWORD param);
	void InitiateModule(void* param[]);
	CM101(BYTE index, BYTE number, void* param[]);
	virtual ~CM101();
private:
	unsigned char* m_tagNmOfChan;
	unsigned char* m_tagChanel[16];
	unsigned char* m_tagWriteCode;
};

#define _M113_ 9
class CM113 :
	public CDevice,
	public CAnalog
{
public:
	void InitiateModule(void* param[]);
	bool ExecuteCommand(DWORD);
	CM113(BYTE index, BYTE number, void* param[]);
	virtual ~CM113();
protected:
	int* m_tagChanN;
	float* m_tagChanel[8];
};


#define _M230_ 15
class CM230 : public CDevice
{
public:
	bool ExecuteCommand(DWORD);
	void InitiateModule(void* param[]);
	CM230(BYTE index, BYTE number, unsigned int basenum, void* param[]);
	virtual ~CM230();

protected:
	unsigned int m_uiBaseTagNumber;
	unsigned char* m_tagCommand;
	int* m_tagCounter[32];
};
```

```
#define _MY01_ 28
class CMY01 : public CDevice
{
public:

	bool ExecuteCommand(DWORD);
	void InitiateModule(void*param[]);
	CMY01(BYTE index, BYTE number,int basenum, void* param[]);
	virtual ~CMY01();

protected:
	int m_uiBaseTagNumber;
	int* m_tagInData[2];
	unsigned char* m_tagRead;
	int* m_tagData;
};
```

Определение методов классов.

```
/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////

CDevice::CDevice(BYTE Index, BYTE NumbFrm, void* mySystem)
{
	m_Index=Index;
	m_NumbFrm=NumbFrm;
	m_pInput = new (BYTE[6]);
	m_pOutput = new (BYTE[255]);
	m_pSystem = mySystem;
	ASSERT( m_pSystem );
};

CDevice::~CDevice()
{
	delete [] m_pInput;
	delete [] m_pOutput;
};


/////////////////////////////////////////////////////////////////////////////
// CAnalog Class
/////////////////////////////////////////////////////////////////////////////

WORD CAnalog::AnalgToDigit (double Analog){

	union{
		WORD dU;
		struct TaU{
		unsigned mnt:11;
		unsigned dip:4;
		unsigned sig:1;
		}aU;
	}Kod;

	if (Analog >= 0) Kod.aU.sig = 0;
	else Kod.aU.sig = 1;
	Kod.aU.dip = CAnalog::Diapazon;
	if (Analog >= 0) Kod.aU.mnt = ((unsigned)(Analog/CAnalog::MantsBit + 1))&0x7ff;
	else Kod.aU.mnt = (~((unsigned)(-Analog/CAnalog::MantsBit + 1)))&0x7ff;

	return Kod.dU;


};
```

```
double CAnalog :: DigitToAnalg (WORD Digital){

	double Analog;
	double m_b;

	union{
		WORD dU;
		struct TaU{
		unsigned mnt:11;
		unsigned dip:4;
		unsigned sig:1;
		}aU;
	}Kod;

	Kod.dU=Digital;
 //Auto choosing of low bit weight
	switch(Kod.aU.dip){
		case 0x08:
			m_b = 0.005;
			break;
		case 0x07:
			m_b = 0.0025;
			break;
		case 0x06:
			m_b = 0.00125;
			break;
		case 0x05:
			m_b = 0.000625;
			break;
		case 0x04:
			m_b = 0.0003125;
			break;
		case 0x03:
			m_b = 0.00015625;
			break;
		case 0x02:
			m_b = 0.000078125;
			break;
		case 0x01:
			m_b = 0.0000390625;
			break;
		case 0x00:
			m_b = 0.00001953125;
			break;
	};

	if (Kod.aU.sig) Analog = - m_b*((~Kod.aU.mnt)&0x7ff);
	else		Analog = m_b*(Kod.aU.mnt);

		return Analog;
};

//////////////////////////////////////////////////////////////////////////////
// CDiscrete Class
//////////////////////////////////////////////////////////////////////////////

WORD CDiscrete::WriteCode(unsigned char Discrete[]){

	WORD wCode = 0;
	WORD tmp=0;
	for (BYTE ctr=0; ctr<WordLength; ctr++)
	{
		tmp=Discrete[ctr];
		wCode = wCode | (tmp<<ctr);
	}

	return wCode;
```

```
};


void CDiscrete::ReadCode(WORD Word, bool* temp){

	for (BYTE Ctr=0; Ctr<WordLength; Ctr++){
		WORD Mask = (WORD)pow(2,Ctr);
		temp[Ctr] =(Word&Mask)>>Ctr;
	}
};

/////////////////////////////////////////////////////////////////////////////
// CM204 Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////
CM204::CM204():CDevice(), CAnalog(0x08, 0.005){
	//default constructor
};

CM204::CM204(BYTE index, BYTE numbinfrm, void** param) : CDevice(index, numbinfrm, param[51]), CAnalog(0x08,
0.005){
 //initiate module inO OPC case
	InitiateModule(param);

};

CM204::~CM204(){

};

//Interfaces of class
void CM204::InitiateModule(void* pData[])
{
 // pData contains all information required to initiate module
  char prefix[40], name[50];
  wsprintf( prefix,"M204_%2.2d.", m_NumbFrm );
 // TimeOut tag
  wsprintf(name, "%sTimeOut", prefix);
  TIntTag* tmout = static_cast<TIntTag*>(pData[0]);
#ifdef _DEBUG
  tmout->value=3000;
#endif
  tmout->setName(name);
  m_tagTimeout = &(tmout->value);
 //ErrCode tag
  wsprintf(name, "%sErrCode", prefix);
  TIntTag* errcode = static_cast<TIntTag*>(pData[1]);
  errcode->setName(name);
  m_tagErrorCode = &(errcode->value);
 //8 Analog Channels
  for (BYTE I=0; I<8; I++){
		wsprintf(name, "%sChanel%2.2d", prefix, I);
		TAnTag* ChanI = static_cast<TAnTag*>(pData[2+I]);
		ChanI->setName( name);
		m_tagChanel[I] = &(ChanI->value);
  };
 //Mask tag
  wsprintf(name, "%sMask", prefix);
  TIntTag* mask = static_cast<TIntTag*>(pData[10]);
  mask->setName(name);
  m_tagMask = &(mask->value);
 //OnOff tag
  wsprintf(name, "%sIsInThread", prefix);
  TBitTag* OnOff = static_cast<TBitTag*>(pData[50]);
  OnOff->setName(name);
```

```
  m_tagOnOff = &OnOff->value;
//!!! !!! !!! !!! !!! !!! !!!
  *m_tagOnOff = 1;
//!!! !!! !!! !!! !!! !!! !!!
  TIntTag* pValid = static_cast<TIntTag*>(pData[52]);
  wsprintf(name, "%sValid", prefix);
  pValid->setName(name);
  m_tagValid = &pValid->value;
 //value of this tag is assign while register this device(CSystem)

 // initialize part of m_pInput
  m_pInput[0] = 6;
  m_pInput[1] = m_NumbFrm;
  m_pInput[2] = 'U';
 //set helpfull variables
  SzInput = 6;
  //m_pSystem!!!

  return;
};

/////////////////////////////////////////////////////////////////////////////
// CM210 Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////

CM210::CM210() : CDevice(), CAnalog(0x08, 0.0005)
{
  // default constructor
};

CM210::CM210(BYTE index, BYTE numbinfrm, UINT TagNmb, void** param) : CDevice(index, numbinfrm, param[51]),
CAnalog(0x08, 0.005)
{
  //initiate module in OPC case
            m_uiBaseTagNumber=TagNmb;
            InitiateModule(param);
};

CM210::~CM210()
{

};

void CM210::InitiateModule (void* pData[]){
  char prefix[40], name[50];
  wsprintf(prefix,"M210_%2.2d.", m_NumbFrm);
 // TimeOut tag
  wsprintf(name, "%sTimeOut", prefix);
  TIntTag* timeout = static_cast<TIntTag*>(pData[0]);
#ifdef _DEBUG
  timeout->value=3000;
#endif
  timeout->setName(name);
  m_tagTimeout = &(timeout->value);
 //ErrCode tag
  wsprintf(name, "%sErrCode", prefix);
  TIntTag* errcode = static_cast<TIntTag*>(pData[1]);
  errcode->setName(name);
  m_tagErrorCode = &(errcode->value);
 // Chanels tags
  for (BYTE I=0; I<16; I++){
              wsprintf(name, "%sChanel%2.2d", prefix, I);
              TAnTag* ChanI = static_cast<TAnTag*>(pData[2+I]);
              ChanI->setName(name);
```

```
            m_tagChanel[I]= &(ChanI->value);
  };
 //m_tagOnOff
  wsprintf(name, "%sIsInThread", prefix);
  TBitTag* OnOff = static_cast<TBitTag*>(pData[50]);
  OnOff->setName(name);
  m_tagOnOff = &OnOff->value;
//!!! !!! !!! !!! !!! !!!
  *m_tagOnOff = 0;
//!!! !!! !!! !!! !!! !!!

  TIntTag* pValid = static_cast<TIntTag*>(pData[52]);
  wsprintf(name, "%sValid", prefix);
  pValid->setName(name);
  m_tagValid = &pValid->value;
 //value assign while initiate CSystem (or test this module)
 // prepare command
  SzInput = 8;
  m_pInput[0] = 11;
  m_pInput[1] = m_NumbFrm;
  m_pInput[2] = 'V';
};



bool CM204::ExecuteCommand(DWORD)
{
  CSystem* system = static_cast<CSystem*>(m_pSystem);
  m_pInput[3] = *m_tagMask;
  m_pInput[4] = 0;
  m_pInput[5] = 0;

  if ( (*m_tagErrorCode = system->SendCommand( m_pInput, m_pOutput, 6, SzOutput, *m_tagTimeout)) )
            return false;  // data isn't correct
  else{
            m204_GetData();
            return true;
  }
};

void CM204::m204_GetData()
{
          BYTE resCtr=0; //counter of bytes we read from result of command
          for (UINT ctr=0; ctr<8; ctr++){
                    int locMask = (((int)pow(2,ctr))&(*m_tagMask))>>ctr;
                    if (locMask){
                              *m_tagChanel[ctr]      =      (float)      DigitToAnalg(      MAKEWORD(      m_pOutput[4+resCtr],
m_pOutput[5+resCtr]) );
                              resCtr++;
                    }
          }
}; //m204_GetData

bool CM210::ExecuteCommand(DWORD param)
{

  //determine number of chanell
          BYTE addr = LOBYTE(LOWORD(param));
          BYTE ChanNmb=m210_GetChan(addr);
          ASSERT( ChanNmb>=0 && ChanNmb<=15 );
  // prepare command for MCS module
          m_pInput[3] = ChanNmb;
          m_pInput[4] = LOBYTE(AnalgToDigit( *(m_tagChanel[ChanNmb])) );
          m_pInput[5] = HIBYTE(AnalgToDigit( *(m_tagChanel[ChanNmb])) );
          m_pInput[6] = 0;
          m_pInput[7] = 0;
  //transmit command and get result through m_pSystem
          CSystem* sys = static_cast<CSystem*>(m_pSystem);
```

```
	if ( !sys->SendCommand( m_pInput, m_pOutput, SzInput, SzOutput, *m_tagTimeout)  )
		return true;
	else
		return false;
}


int CM210::m210_GetChan(UINT addr)
{
  return (addr-m_uiBaseTagNumber);
}

/////////////////////////////////////////////////////////////////////////////
// CMCSFrame Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////
CMCSFrame::CMCSFrame(void** param) : CDevice(param[51])
{
	InitiateModule(param);
}

CMCSFrame::~CMCSFrame()
{

}

void CMCSFrame::InitiateModule(void *param[])
{
	CSystem* pSys = static_cast<CSystem*>(m_pSystem);

	char prefix[50], name[100], port[20];
	switch (pSys->m_PortNumber){
	case ISCOM1:
		wsprintf(port,"COM1");
		break;
	case ISCOM2:
		wsprintf(port, "COM2");
		break;
	case ISCOM3:
		wsprintf(port,"COM3");
		break;
	case ISCOM4:
		wsprintf(port,"COM4");
		break;
	case ISPPRT:
		wsprintf(port,"PPRT");
		break;
	case ISLPT:
		wsprintf(port, "LPT");
		break;
	default:
		wsprintf(port, "Unknown!!!");
	};

	wsprintf(prefix, "Frame_on_%s.", port);
  // timeout of operations and error tag
	TIntTag* tmout = static_cast<TIntTag*>(param[0]);
	wsprintf(name, "%sTimeOut", prefix);
	tmout->setName(name);
	m_tagTimeout = &tmout->value;
#ifdef _DEBUG
	*m_tagTimeout = 3000;
#endif
	TIntTag* errcde = static_cast<TIntTag*>(param[1]);
	wsprintf(name, "%sErrCode", prefix);
```

```
		errcde->setName(name);
		m_tagErrorCode=&errcde->value;


	//tags specific for frame object(in OPC server case)
		TIntTag* pCommand = static_cast<TIntTag*>(param[2]);
		wsprintf(name, "%sCOMMAND", prefix);
		pCommand->setName( name);
		m_tagCommandChan=&pCommand->value;

		TBitTag* pState = static_cast<TBitTag*>(param[3]);
		wsprintf(name,"%sSTATE", prefix);
		pState->setName(name);
		m_tagOnOff=&pState->value;

		TIntTag* pValid = static_cast<TIntTag*>(param[52]);
		wsprintf(name, "%sValid", prefix);
		pValid->setName(name);
		m_tagValid = &pValid->value;
}


bool CMCSFrame::ExecuteCommand(DWORD param)
{
	//detect type of action
		int operation=*m_tagCommandChan;

		CSystem* pSys = static_cast<CSystem*>(m_pSystem);

		switch(operation){
		case MCS_RESTART:
			EnterCriticalSection(pSys->m_pReadWriteCritical);
			 pSys->m_sysRestart();
			LeaveCriticalSection(pSys->m_pReadWriteCritical);
			break;
		case MCS_THREADON:
			pSys->m_sysStartThreads();
			*m_tagOnOff=1;
			break;
		case MCS_THREADOFF:
			pSys->m_sysStopThreads();
			*m_tagOnOff=0;
			break;
		case MCS_INITAGAIN:
			EnterCriticalSection(pSys->m_pReadWriteCritical);
			 pSys->mcsInitAgain();
			LeaveCriticalSection(pSys->m_pReadWriteCritical);
			break;
		}
		return true;
}


bool CDevice::IsInTheThread()
{
		return ((bool)(*m_tagOnOff));
}

CDevice::CDevice(void *pSystem)
{
		m_pSystem = pSystem;
		ASSERT(m_pSystem);
}

void CDevice::TestModule()
{
		if (*m_tagValid == MODULE_TEST){
		CSystem* pSys = static_cast<CSystem*>(m_pSystem);
```

```
		// test of module is general operation for all modules
			BYTE* command = new BYTE[6];
			BYTE* result = new BYTE[255];
			ZeroMemory(command, 6);
			ZeroMemory(result, 255);

			command[0] = m_Index;
			command[1] = m_NumbFrm;
			command[2] = 'T';

			int err = *m_tagErrorCode = pSys->SendCommand(command, result, 6, SzOutput,3000);


			*m_tagValid = ((int)result[5])<<8 + result[4];

			delete [] command;
			delete [] result;
		} //if (*m_tagValid == MODULE_TEST)
}

/////////////////////////////////////////////////////////////////////////////
// CM102 Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////

CM102::CM102(BYTE index, BYTE number, void* param[]) : CDevice(index, number, param[51]), CDiscrete()
{
		SetPrefix();
		InitiateModule(param);
}

CM102::~CM102()
{

}

void CM102::InitiateModule(void* param[])
{
  char name[100];


  TIntTag* tmout = static_cast<TIntTag*>(param[0]);
  wsprintf(name, "%sTimeOut", prefix);
  tmout->setName(name);
  m_tagTimeout=&tmout->value;
#ifdef _DEBUG
  *m_tagTimeout=3000;
#endif

  TIntTag* errcde = static_cast<TIntTag*>(param[1]);
  wsprintf(name, "%sErrCode", prefix);
  errcde->setName(name);
  m_tagErrorCode = &errcde->value;

  TBitTag* ChanI;
  for (BYTE Ctr=0; Ctr<16; Ctr++){
		ChanI = static_cast<TBitTag*>(param[2+Ctr]);
		wsprintf(name, "%sChanel%2.2d", prefix,Ctr);
		ChanI->setName(name);
		m_tagChanel[Ctr] = &ChanI->value;
  };

  TBitTag* write = static_cast<TBitTag*>(param[18]);
  wsprintf(name,"%sSendCode", prefix);
  write->setName(name);
```

```
  m_tagWrteCode = &write->value;

  TBitTag* thread = static_cast<TBitTag*>(param[50]);
  wsprintf(name, "%sIsInThread", prefix);
  thread->setName(name);
  m_tagOnOff=&thread->value;
  *m_tagOnOff = 0;

  TIntTag* valid = static_cast<TIntTag*>(param[52]);
  wsprintf(name,"%sValid", prefix);
  valid->setName(name);
  m_tagValid = &valid->value;

}

bool CM102::ExecuteCommand(DWORD)
{
  //this operation intialized by OPC client
  //sertting to On value of BitTag
  // when this operaton complete set this tag to Off state
		CSystem* pSys = static_cast<CSystem*>(m_pSystem);

		unsigned char dCode [16];
		for (BYTE ctr=0; ctr<16; ctr++) dCode[ctr] = *m_tagChanel[ctr];
		WORD wCode = WriteCode( dCode );

		m_pInput[0] = m_Index;
		m_pInput[1] = m_NumbFrm;
		m_pInput[2] = 'C';
		m_pInput[3] = 0;
		m_pInput[4] = LOBYTE(wCode);
		m_pInput[5] = HIBYTE(wCode);

		*m_tagErrorCode=pSys->SendCommand(m_pInput, m_pOutput, 6, SzOutput, *m_tagTimeout);
		*m_tagWrteCode = 0;

		return true;
}

/////////////////////////////////////////////////////////////////////////////
// CM103 Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////
CM103::CM103(BYTE index, BYTE number, void* param[]) : CM102(index, number, param)
{
		this->SetPrefix();
		InitiateModule(param);
}

CM103::~CM103()
{

}

/////////////////////////////////////////////////////////////////////////////
// CM201 Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////

CM201::CM201()
{
```

```
}

CM201::~CM201()
{

}

void CM201::InitiateModule(void *param[])
{
	char prefix[50], name[100];
	wsprintf(prefix, "M201_%2.2d.", m_NumbFrm);

	TIntTag* tmout = static_cast<TIntTag*>(param[0]);
	wsprintf(name, "%sTimeOut", prefix);
	tmout->setName(name);
	m_tagTimeout=&tmout->value;
#ifdef _DEBUG
	*m_tagTimeout = 3000;
#endif

	TIntTag* errcde = static_cast<TIntTag*>(param[1]);
	wsprintf(name,"%sErrCode",prefix);
	errcde->setName(name);
	m_tagErrorCode=&errcde->value;

	TBitTag* ChanI;
	for (BYTE ctr=0; ctr<16; ctr++){
		ChanI = static_cast<TBitTag*>(param[2+ctr]);
		wsprintf(name, "%sChanel0_%2.2d", prefix, ctr);
		ChanI->setName(name);
		m_tagChanel0[ctr] = &ChanI->value;
	}

	for (ctr=0; ctr<16; ctr++){
		ChanI = static_cast<TBitTag*>(param[18+ctr]);
		wsprintf(name, "%sChanel1_%2.2d", prefix, ctr);
		ChanI->setName(name);
		m_tagChanel1[ctr] = &ChanI->value;
	}

	TBitTag* read = static_cast<TBitTag*>(param[34]);
	wsprintf(name, "%sReadCode", prefix);
	read->setName(name);
	m_tagReadCode = &read->value;


	TBitTag* thread  = static_cast<TBitTag*>(param[50]);
	wsprintf(name, "%sIsInThread", prefix);
	thread->setName(name);
	m_tagOnOff =&thread->value;
	*m_tagOnOff = 0;

	TIntTag* valid = static_cast<TIntTag*>(param[52]);
	wsprintf(name,"%sValid",prefix);
	valid->setName(name);
	m_tagValid = &valid->value;

  //initialize command
	m_pInput[0] = m_Index;
	m_pInput[1] = m_NumbFrm;
	m_pInput[2] = 'D';
	m_pInput[3] = 0;


}

bool CM201::ExecuteCommand(DWORD)
{
```

```
	CSystem* pSys = static_cast<CSystem*>(m_pSystem);

	int err = *m_tagErrorCode = pSys->SendCommand(m_pInput, m_pOutput, 4, SzOutput, *m_tagTimeout);
	if (err==OKCODE){
		WORD wCode0 = MAKEWORD(m_pOutput[4], m_pOutput[5]);
		WORD wCode1 = MAKEWORD(m_pOutput[6], m_pOutput[7]);
		bool dCode0[16];
		bool dCode1[16];
		ReadCode(wCode0, dCode0);
		ReadCode(wCode1, dCode1);
		for (BYTE ctr=0; ctr<16; ctr++) *m_tagChanel0[ctr] = dCode0[ctr];
		for (ctr=0; ctr<16; ctr++) *m_tagChanel1[ctr] = dCode1[ctr];
		*m_tagReadCode = 0;
		return true;
	}
	*m_tagReadCode = 0;
	return false;
}

CM201::CM201(BYTE index, BYTE number, void *param[]) : CDevice(index, number, param[51]), CDiscrete()
{
	InitiateModule(param);
}


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CM202::CM202(BYTE index, BYTE number, void* param[]) : CDevice(index, number, param[51]), CDiscrete(8)
{
	InitiateModule(param);
}

CM202::~CM202()
{

}


void CM202::InitiateModule(void *param[])
{
	char prefix[50], name[100];
	wsprintf(prefix, "M202_%2.2d.", m_NumbFrm);

	TIntTag* tmout=static_cast<TIntTag*>(param[0]);
	wsprintf(name,"%sTimeOut",prefix);
	tmout->setName(name);
	m_tagTimeout=&tmout->value;
#ifdef _DEBUG
	*m_tagTimeout=3000;
#endif

	TIntTag* errcde=static_cast<TIntTag*>(param[1]);
	wsprintf(name,"%sErrCode",prefix);
	errcde->setName(name);
	m_tagErrorCode=&errcde->value;

	TBitTag* ChanI;
	for (BYTE ctr=0; ctr<8; ctr++){
		ChanI = static_cast<TBitTag*>(param[2+ctr]);
		wsprintf(name,"%sChanel%2.2d", prefix, ctr);
		ChanI->setName(name);
		m_tagChanel[ctr] = &ChanI->value;
	}

	TBitTag* write = static_cast<TBitTag*>(param[10]);
	wsprintf(name,"%sWriteCode", prefix);
```

```
		write->setName(name);
		m_tagWriteCode=&write->value;


		TBitTag* thread = static_cast<TBitTag*>(param[50]);
		wsprintf(name,"%sIsInThread", prefix);
		thread->setName(name);
		m_tagOnOff=&thread->value;

		TIntTag* valid = static_cast<TIntTag*>(param[52]);
		wsprintf(name,"%sValid", prefix);
		valid->setName(name);
		m_tagValid=&valid->value;



	//prepare command
		m_pInput[0]=m_Index;
		m_pInput[1]=m_NumbFrm;
		m_pInput[2]='C';
		m_pInput[3]=0;



}

bool CM202::ExecuteCommand(DWORD)
{
		CSystem* pSys = static_cast<CSystem*>(m_pSystem);
		unsigned char dCode[8];
		for (BYTE ctr=0; ctr<8; ctr++)
			dCode[ctr] = *m_tagChanel[ctr];

		WORD wCode = WriteCode(dCode);

		m_pInput[4] = LOBYTE(wCode);
		m_pInput[5] = 0;

		int err = *m_tagErrorCode = pSys->SendCommand(m_pInput, m_pOutput, 6, SzOutput, *m_tagTimeout);
		*m_tagWriteCode=0;
		if (err!=OKCODE)
			return false;
		else
			return true;
}

/////////////////////////////////////////////////////////////////////////////
// CM203 Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////

CM203::CM203(BYTE index, BYTE number, void* param[]) : CDevice(index, number, param[51]), CDiscrete()
{
		InitiateModule(param);
}

CM203::~CM203()
{

}

void CM203::InitiateModule(void *param[])
{
		char prefix[50], name[100];
		wsprintf(prefix,"M203_%2.2d.", m_NumbFrm);
```

```
		preInitModule(prefix, param);

		TBitTag* ChanI;
		for (BYTE ctr=0; ctr<16; ctr++){
			ChanI = static_cast<TBitTag*>(param[2+ctr]);
			wsprintf(name, "%sChanel0_%2.2d",prefix, ctr);
			ChanI->setName(name);
			m_tagChanel0[ctr]=&ChanI->value;
		}

		for (ctr=0; ctr<16; ctr++){
			ChanI = static_cast<TBitTag*>(param[18+ctr]);
			wsprintf(name, "%sChanel1_%2.2d",prefix, ctr);
			ChanI->setName(name);
			m_tagChanel1[ctr]=&ChanI->value;
		}

		TBitTag* write = static_cast<TBitTag*>(param[34]);
		wsprintf(name, "%sWriteCode", prefix);
		write->setName(name);
		m_tagWriteCode=&write->value;

		TIntTag* chan = static_cast<TIntTag*>(param[35]);
		wsprintf(name,"%sNumberOfRegister", prefix);
		chan->setName(name);
		m_tagChanN=&chan->value;
		*m_tagChanN = 0;
	//prepare command
		m_pInput[0] = m_Index;
		m_pInput[1] = m_NumbFrm;
		m_pInput[2] = 'C';

};

bool CM203::ExecuteCommand(DWORD)
{
		CSystem* pSys=static_cast<CSystem*>(m_pSystem);
		WORD wCode;
		unsigned char dCode[16];
		if (*m_tagChanN==0)
			for (wCode=0; wCode<16; wCode++)
				dCode[wCode] = *m_tagChanel0[wCode];
		else
			for (wCode=0; wCode<16; wCode++)
				dCode[wCode] = *m_tagChanel1[wCode];

		wCode = WriteCode(dCode);
		m_pInput[3] = *m_tagChanN;
		m_pInput[4] = LOBYTE(wCode);
		m_pInput[5] = HIBYTE(wCode);

		int err = *m_tagErrorCode=pSys->SendCommand(m_pInput, m_pOutput, 6, SzOutput, *m_tagTimeout);
		*m_tagWriteCode=0;
		if (err!=OKCODE)
			return false;
		else
			return true;
}

void CDevice::preInitModule(const char* prefix, void *param[])
{
	char name[100];
	// common for all modules operations
		TIntTag* tmout = static_cast<TIntTag*>(param[0]);
		wsprintf(name, "%sTimeOut", prefix);
		tmout->setName(name);
		m_tagTimeout=&tmout->value;
#ifdef _DEBUG
```

```
	*m_tagTimeout=3000;
#endif
	TIntTag* errcde = static_cast<TIntTag*>(param[1]);
	wsprintf(name,"%sErrCode",prefix);
	errcde->setName(name);
	m_tagErrorCode=&errcde->value;

	TBitTag* thread = static_cast<TBitTag*>(param[50]);
	wsprintf(name, "%sIsInThread", prefix);
	thread->setName(name);
	m_tagOnOff=&thread->value;
	*m_tagOnOff=0;

	TIntTag* valid = static_cast<TIntTag*>(param[52]);
	wsprintf(name,"%sValid", prefix);
	valid->setName(name);
	m_tagValid=&valid->value;
}

//////////////////////////////////////////////////////////////////////
// CM101 Class
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CM101::CM101(BYTE index, BYTE number, void* param[]) :
CDevice( index, number, param[51]),
CDiscrete()
{
	InitiateModule(param);
}

CM101::~CM101()
{

}


//////////////////////////////////////////////////////////////////////
// CM113 Class
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CM113::CM113(BYTE index, BYTE number, void* param[]) : CDevice(index, number, param[51]), CAnalog()
{
	InitiateModule(param);
}

CM113::~CM113()
{

}

void CM113::InitiateModule(void *param[])
{
	char prefix[50], name[100];
	wsprintf(prefix, "M113_%2.2d.", m_NumbFrm);
	preInitModule(prefix, param);

	TAnTag* ChanI;
	for (BYTE ctr=0; ctr<8; ctr++){
		ChanI = static_cast<TAnTag*>(param[2+ctr]);
		wsprintf(name,"%sChanel%2.2d",prefix,ctr);
```

```
		ChanI->setName(name);
		m_tagChanel[ctr] = &ChanI->value;
	};

	TIntTag* chan = static_cast<TIntTag*>(param[10]);
	wsprintf(name,"%sNumberOfChanel", prefix);
	chan->setName(name);
	m_tagChanN=&chan->value;

 //prepare command
	m_pInput[0] = m_Index;
	m_pInput[1] = m_NumbFrm;
	m_pInput[2] = 'U';
}

bool CM113::ExecuteCommand(DWORD)
{
	int ChanN = *m_tagChanN;
	if (ChanN<0 || ChanN>7)
		return false;

	CSystem* pSys = static_cast<CSystem*>(m_pSystem);

	m_pInput[3] = ChanN;
	int err = *m_tagErrorCode=pSys->SendCommand(m_pInput, m_pOutput, 4, SzOutput, *m_tagTimeout);
	if (err==OKCODE){
		WORD wCode = MAKEWORD(m_pOutput[4], m_pOutput[5]);
		*m_tagChanel[ChanN] = DigitToAnalg( wCode);
		return true;
	}
	else
		return false;
}

/////////////////////////////////////////////////////////////////////////////
// CM230 Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////

CM230::CM230(BYTE index, BYTE number, unsigned int basenum, void* param[]) : CDevice(index, number, param[51])
{
	m_uiBaseTagNumber = basenum;
	InitiateModule(param);
}

CM230::~CM230()
{

}

void CM230::InitiateModule(void *param[])
{
	char prefix[50], name[100];
	wsprintf(prefix, "M230_%2.2d.", m_NumbFrm);
	preInitModule(prefix,param);

	TIntTag* CountI;
	for (BYTE ctr=0; ctr<32; ctr++){
		CountI = static_cast<TIntTag*>(param[2+ctr]);
		wsprintf(name, "%sCounter%2.2d",prefix,ctr);
		CountI->setName(name);
		m_tagCounter[ctr] = &CountI->value;
	};

	TBitTag* command = static_cast<TBitTag*>(param[34]);
```

```
		wsprintf(name,"%sReadCounters",prefix);
		command->setName(name);
		m_tagCommand = &command->value;
	//prepare command
		m_pInput[0] = m_Index;
		m_pInput[1] = m_NumbFrm;
}

bool CM230::ExecuteCommand(DWORD number)
{
		bool command = *m_tagCommand;
		int err = ~OKCODE;
		CSystem* pSys = static_cast<CSystem*>(m_pSystem);

		if (command){
			m_pInput[2] = 'D';
			m_pInput[3] = 0;
			err = *m_tagErrorCode=pSys->SendCommand(m_pInput, m_pOutput, 4, SzOutput, *m_tagTimeout);
			if (err==OKCODE)
				for (BYTE ctr=0; ctr<32; ctr+=2)
					*m_tagCounter[ctr] = MAKEWORD(m_pOutput[4+ctr], m_pOutput[5+ctr]);
			*m_tagCommand=0;
		}
		else{
			WORD wCode;
			wCode = *m_tagCounter[number-m_uiBaseTagNumber];
			m_pInput[2] = 'C';
			m_pInput[3] = (number - m_uiBaseTagNumber);
			m_pInput[4] = LOBYTE(wCode);
			m_pInput[5] = HIBYTE(wCode);
			err = *m_tagErrorCode = pSys->SendCommand(m_pInput, m_pOutput, 6, SzOutput, *m_tagTimeout);
		}

		if (err==OKCODE)
			return true;
		else
			return false;
}

/////////////////////////////////////////////////////////////////////////////
// CMY01 Class
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////////////////

CMY01::CMY01(BYTE index, BYTE number,int basenum, void* param[]) : CDevice(index, number, param[51])
{
		m_uiBaseTagNumber=basenum;
		InitiateModule(param);
}

CMY01::~CMY01()
{

}

void CMY01::InitiateModule(void *param[])
{
		char prefix[50], name[100];
		wsprintf(prefix, "MY01_%2.2d.", m_NumbFrm);
		preInitModule(prefix, param);

		TIntTag* data = static_cast<TIntTag*>(param[4]);
		wsprintf(name, "%sOutputData", prefix);
		data->setName(name);
		m_tagData = &data->value;
```

```
		data = static_cast<TIntTag*>(param[2]);
		wsprintf(name, "%sInputData0", prefix);
		data->setName(name);
		m_tagInData[0] = &data->value;

		data = static_cast<TIntTag*>(param[3]);
		wsprintf(name, "%sInputData1", prefix);
		data->setName(name);
		m_tagInData[1] = &data->value;

		TBitTag* read=static_cast<TBitTag*>(param[5]);
		wsprintf(name,"%sReadData",prefix);
		read->setName(name);
		m_tagRead = &read->value;
	//prepare command
		m_pInput[0] = m_Index;
		m_pInput[1] = m_NumbFrm;

}

bool CMY01::ExecuteCommand(DWORD number)
{
		bool comm = *m_tagRead;
		int err = !OKCODE;
		CSystem* pSys = static_cast<CSystem*>(m_pSystem);
		if (comm){
			m_pInput[2] = 'D';
			m_pInput[3] = 0;
			err = *m_tagErrorCode=pSys->SendCommand(m_pInput, m_pOutput, 4, SzOutput, *m_tagTimeout);
			if (err==OKCODE)
				*m_tagData=MAKEWORD(m_pOutput[4], m_pOutput[5]);
			*m_tagRead = 0;
		}
		else{
			int chann=number-m_uiBaseTagNumber;
			if (chann<0 || chann>1)
				return false;
			m_pInput[2] = 'C';
			m_pInput[3] = chann;
			m_pInput[4] = LOBYTE(*m_tagInData[chann]);
			m_pInput[5] = HIBYTE(*m_tagInData[chann]);

			err = *m_tagErrorCode =pSys->SendCommand(m_pInput, m_pOutput, 6, SzOutput, *m_tagTimeout);
		};

		if (err==OKCODE)
			return true;
		else
			return false;
}

void CM101::InitiateModule(void *param[])
{
		char prefix[50], name[100];
		wsprintf(prefix,"M101__%d",m_NumbFrm);
		preInitModule(prefix, param);


}

bool CM101::ExecuteCommand(DWORD param)
{
		return false;

}

// System.cpp: implementation of the CSystem class.
```

```
//
//////////////////////////////////////////////////////////////////////

#include <Afx.h>
#include <windows.h>
#include <process.h>
#include "System.h"
#include "Module.h"
#include "dataserv.h"
#include "CArray.h"
#include "CQuerry.h"
#include "MCS\\Sendbyte.h"



extern TIntTag IntTag[];
extern TAnTag  AnTag[];
extern TBitTag BitTag[];

//////////////////////////////////////////////////////////////////////////////////////////
//Global counters of registered tags(use it when register devices in CSystem::mcsRegister)
//
//////////////////////////////////////////////////////////////////////////////////////////
int IntTagCtr;
int BitTagCtr;
int AngTagCtr;


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
CSystem::CSystem(BYTE portNmb)
{
  m_PortNumber = portNmb;
  m_pTableOfResources = new BYTE[255];

  ASSERT(m_pTableOfResources!=NULL);

  ZeroMemory(m_pTableOfResources, 255);

 //!!!IMPORTANT: intialize querrys before calling to mcsRegister() !!!
  m_pCycle = new CQuerry<CDevice>;
  m_pQuerry = new CQuerry<CSystem::TDeviceStruct>;

 //browse scope of tags of OPC server
  mcsStart();

  m_pReadWriteCritical=new (CRITICAL_SECTION);
  ASSERT(m_pReadWriteCritical);
  InitializeCriticalSection( m_pReadWriteCritical );

 // start threads (suspend CycleThread)
  m_hCycleThread = (HANDLE) _beginthread( CycleThread, 0, (void*)this);

//  SuspendThread(m_hCycleThread);
  m_hQuerryThread = (HANDLE) _beginthread( QuerryThread, 0 ,(void*)this);

 //set flag of MCS starting in false state
  Started = false;
}

CSystem::~CSystem()
{
  if (m_pTableOfResources!=NULL)
              delete [] m_pTableOfResources;
}

//////////////////////////////////////////////////////////////////////
```

```
// CInterActScope Class
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CInterActScope::CInterActScope()
{
   for (UINT ctr=0; ctr<6; ctr++)
                        m_pSystem[ctr] = NULL;
  //set global counters of reserved tags to zero
            IntTagCtr=BitTagCtr=AngTagCtr=0;
  //create set of System objects each object associated with port
            m_pSystem[0] = new CSystem(ISCOM1);
//          m_pSystem[1] = new CSystem(ISCOM2);
//          m_pSystem[2] = new CSystem(ISCOM3);
//          m_pSystem[3] = new CSystem(ISCOM4);
//          m_pSystem[4] = new CSystem(ISPPRT);
//          m_pSystem[5] = new CSystem(ISLPT);
}

CInterActScope::~CInterActScope()
{
            for (BYTE ctr=0; ctr<6; ctr++)
                        if (m_pSystem[ctr]!=NULL)
                                    delete m_pSystem[ctr];
}


bool CSystem::mcsInitiate()
{

  BYTE mcsInit[6];

  mcsInit[0] = 0;
  mcsInit[1] = 0;
  mcsInit[2] = 'L';
  mcsInit[3] = 0;

  int errcode;

  errcode = SendCommand(mcsInit, m_pTableOfResources, 4, m_uiSzResTab, 3000);

  if (errcode == OKCODE)
             return true;
  else
             return false;
}
//////////////////////////////////////////////////////////////////////////////////////////
// IMPORTANT: to provide operation of MCS restart witohout loosing tags in OPC
//          we should store tags, So use auto_counter's(see AutoCtr.h)
//
//////////////////////////////////////////////////////////////////////////////////////////


bool CSystem::mcsRegister (){
            void* param[100];
  // use global counters of tags (XxxTagCtr)
            UINT ModEmount = ( UINT(m_pTableOfResources[3]*256+m_pTableOfResources[2] - 4))/6 - 1; // emount of modules
in MCS
  // get information about each module and browse a scope of tags in OPC

            for (UINT Ctr=0; Ctr<ModEmount*6; Ctr+=6){

   //index of module
             BYTE moduleInd = m_pTableOfResources[4+Ctr];
            //number in frame
```

```
    BYTE moduleNmb = m_pTableOfResources[5+Ctr];
   //state of module
    WORD moduleStat = MAKEWORD(m_pTableOfResources[8+Ctr], m_pTableOfResources[9+Ctr]);



    switch(moduleInd){
    case _M204_:
            {

                    param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                    param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                    for (BYTE ctr=0; ctr<8; ctr++)
                            param[2+ctr] = static_cast<void*>(&AnTag[AngTagCtr++]);
                    param[10] = static_cast<void*>(&IntTag[IntTagCtr++]);

                    param[50] = static_cast<void*>(&BitTag[BitTagCtr++]);
                    param[51] = static_cast<void*>(this);
// state of module (result of startup test)
                    IntTag[IntTagCtr].value = moduleStat;
                    param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                    // pass "command" number as index of array element
                    // for this module "command" tag number is number of Mask tag
                    //(i.e. number of last tag)
                    CM204* pM204 = new CM204(moduleInd, moduleNmb, param);
                    IntDevices.Add( pM204, IntTagCtr-2 ); //mask tag
                    TstDevices.Add( pM204, IntTagCtr-1 ); //valid tag
                    m_pCycle->Add( pM204);
                    break;
            } //_M204_
    case _M210_:
            {

                    param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                    param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                    for ( BYTE ctr=0; ctr<16; ctr++)
                            param[2+ctr] = static_cast<void*>(&AnTag[AngTagCtr++]);


                    param[50]  = static_cast<void*>(&BitTag[BitTagCtr++]);
                    param[51] =static_cast<void*>(this);
            // state of this device(result of startup test)
                    IntTag[IntTagCtr].value = moduleStat;
                    param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                    AngTagCtr-=16;
                    CM210* pM210 = new CM210(moduleInd, moduleNmb, AngTagCtr, param);

                    TstDevices.Add( pM210, IntTagCtr-1);
                    for ( ctr=0; ctr<16; ctr++)
                            AngDevices.Add (pM210, AngTagCtr++);

                    break;
            } //_M210_

    case _M102_:
            {

                    param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                    param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                    for (BYTE ctr=0; ctr<16; ctr++)
                            param[2+ctr] = static_cast<void*>(&BitTag[BitTagCtr++]);
                    param[18] = static_cast<void*>(&BitTag[BitTagCtr++]);
                    param[50] = static_cast<void*>(&BitTag[BitTagCtr++]);
                    param[51] = static_cast<void*>(this);
                    IntTag[IntTagCtr].value=moduleStat;
                    param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);
```

```
                CM102* pM102 = new CM102(moduleInd, moduleNmb, param);
                TstDevices.Add(pM102, IntTagCtr-1);
                BitDevices.Add(pM102, BitTagCtr-2);
                break;
        }// _M102_
    case _M103_:
        {

                param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                for (BYTE ctr=0; ctr<16; ctr++)
                        param[2+ctr] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[18] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[50] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[51] = static_cast<void*>(this);
                IntTag[IntTagCtr].value=moduleStat;
                param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                CM103* pM103 = new CM103(moduleInd, moduleNmb, param);
                TstDevices.Add(pM103, IntTagCtr-1);
                BitDevices.Add(pM103, BitTagCtr-2);
                break;
        } //_M103_
    case _M201_:
        {

                param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                for (BYTE ctr=0; ctr<32; ctr++)
                        param[2+ctr] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[34] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[50] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[51] = static_cast<void*>(this);
                IntTag[IntTagCtr].value=moduleStat;
                param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                CM201* pM201 = new CM201(moduleInd, moduleNmb, param);
                TstDevices.Add(pM201, IntTagCtr-1);
                BitDevices.Add(pM201, BitTagCtr-2);
#ifndef _SHORT_
                m_pCycle->Add( pM201);
#endif
                break;
        } //_M201_
    case _M202_:
        {

                param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                for (BYTE ctr=0; ctr<9; ctr++)
                        param[2+ctr] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[10] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[50] = static_cast<void*>(&BitTag[BitTagCtr++]);
                param[51] = static_cast<void*>(this);
                IntTag[IntTagCtr].value=moduleStat;
                param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                CM202* pM202 = new CM202(moduleInd, moduleNmb, param);
                TstDevices.Add(pM202, IntTagCtr-1);
                BitDevices.Add(pM202, BitTagCtr-2);
                break;
        }//_M202_

    case _M203_:
        {

                param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
```

```
                        param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        for (BYTE ctr=0; ctr<32; ctr++)
                                param[2+ctr] = static_cast<void*>(&BitTag[BitTagCtr++]);
                        param[34] = static_cast<void*>(&BitTag[BitTagCtr++]);
                        param[35] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        param[50] = static_cast<void*>(&BitTag[BitTagCtr++]);
                        param[51] = static_cast<void*>(this);
                        IntTag[IntTagCtr].value=moduleStat;
                        param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                        CM203* pM203 = new CM203(moduleInd, moduleNmb, param);
                        TstDevices.Add(pM203, IntTagCtr-1);
                        BitDevices.Add(pM203, BitTagCtr-2);
#ifndef _SHORT_
                        m_pCycle->Add(pM203);
#endif
                        break;
                }//_M203_
/*
        case _M101_:
                {

                        param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        for (BYTE ctr=0; ctr<16; ctr++)
                                param[2+ctr] = static_cast<void*>(&BitTag[BitTagCtr++]);
                        param[18] = static_cast<void*>(&BitTag[BitTagCtr++]);
                        param[50] = static_cast<void*>(&BitTag[BitTagCtr++]);
                        param[51] = static_cast<void*>(this);
                        IntTag[IntTagCtr].value=moduleStat;
                        param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                        CM101* pM101 = new CM101(moduleInd, moduleNmb, param);
                        TstDevices.Add(pM101, IntTagCtr-1);
                        BitDevices.Add(pM101, BitTagCtr-2);
                        break;
                }//_M101_*/
        case _M113_:
                {

                        param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        for (BYTE ctr=0; ctr<8; ctr++)
                                param[2+ctr] = static_cast<void*>(&AnTag[AngTagCtr++]);
                        param[10] = static_cast<void*>(&IntTag[IntTagCtr++]);

                         param[50] = static_cast<void*>(&BitTag[BitTagCtr++]);
                         param[51] = static_cast<void*>(this);
    // state of module (result of startup test)
                         IntTag[IntTagCtr].value = moduleStat;
                         param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                        // pass "command" number as index of array element
                        // for this module "command" tag number is number of Mask tag
                        //(i.e. number of last tag)
                         CM113* pM113 = new CM113(moduleInd, moduleNmb, param);
                         IntDevices.Add( pM113, IntTagCtr-2 ); //mask tag
                         TstDevices.Add( pM113, IntTagCtr-1 ); //valid tag
                         break;
                };//_M113_
        case _M230_:
                {
                        param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        for ( BYTE ctr=0; ctr<32; ctr++)
                                param[2+ctr] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        param[34] = static_cast<void*>(&BitTag[BitTagCtr++]);
```

```
                        param[50]  = static_cast<void*>(&BitTag[BitTagCtr++]);
                        param[51] =static_cast<void*>(this);
                // state of this device(result of startup test)
                        IntTag[IntTagCtr].value = moduleStat;
                        param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                        CM230* pM230 = new CM230(moduleInd, moduleNmb, IntTagCtr-33, param);

                        TstDevices.Add( pM230, IntTagCtr-1);
                        IntTagCtr-=33;
                        for ( ctr=0; ctr<32; ctr++)
                                IntDevices.Add (pM230, IntTagCtr++);
                        IntTagCtr++;
                        BitDevices.Add(pM230, BitTagCtr-2);
                        break;
                }
        case _MY01_:
                {
                        param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        for (BYTE ctr=0; ctr<2; ctr++)
                                param[2+ctr] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        param[4] = static_cast<void*>(&IntTag[IntTagCtr++]);
                        param[5] = static_cast<void*>(&BitTag[BitTagCtr++]);

                        param[50]  = static_cast<void*>(&BitTag[BitTagCtr++]);
                        param[51] =static_cast<void*>(this);
                // state of this device(result of startup test)
                        IntTag[IntTagCtr].value = moduleStat;
                        param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);

                        CMY01* pMY01 = new CMY01(moduleInd, moduleNmb, IntTagCtr-4, param);

                        TstDevices.Add( pMY01, IntTagCtr-1);
                        IntDevices.Add( pMY01, IntTagCtr-4);
                        IntDevices.Add( pMY01, IntTagCtr-3);
                        BitDevices.Add( pMY01, BitTagCtr-2);
                        break;

                }//_MY01_
        } //switch(moduleIndex)
    }//for (Ctr ..)
    return true;
}

WORD CSystem::SendCommand(BYTE* inputarray, BYTE* outputarray,UINT SzInput, UINT &MxSzOut, UINT TmOut)
{
  MxSzOut = 255;

    int snderr; // error code of sendbyt() function call
    snderr = sendbyt(ISOPEN, m_PortNumber, inputarray, SzInput, outputarray, MxSzOut, CBR_9600,1000);

    if (snderr==OKCODE)
        snderr = sendbyt(ISWAITEND, m_PortNumber, NULL, 0, NULL, MxSzOut, 0, TmOut);
    else
        return snderr;

    if (snderr==OKCODE)
        snderr = sendbyt(ISCLOSE, m_PortNumber, NULL, 0, NULL, MxSzOut, 0, 2000);
    else
        return snderr;

    if (snderr == OKCODE)
        return snderr;
    else
        return snderr;
}
```

```
bool CInterActScope::onAnalogTag(DWORD param)
{

	//just while debugging
	for (int ctr=0; ctr<5; ctr++)
		if (m_pSystem[ctr]!=NULL)
			return (m_pSystem[ctr]->onAnalog(param));
}

bool CInterActScope::onIntegerTag(DWORD param)
{
	for (int ctr=0; ctr<5; ctr++)
		if (m_pSystem[ctr]!=NULL)
			return (m_pSystem[ctr]->onInteger(param));
}

bool CInterActScope::onBitTag(DWORD param)
{
	for (int ctr=0; ctr<5; ctr++)
		if (m_pSystem[ctr]!=NULL)
			return (m_pSystem[ctr]->onBit(param));
}

bool CSystem::onAnalog(DWORD param)
{
	/*
	 for debug only: try to use sparse array of devices
	*/
  //1. first parameter will be the number of tag
	UINT TagNmb = LOBYTE(LOWORD(param));

	CDevice* tmpDevice = AngDevices[TagNmb];
	if (tmpDevice==NULL)
		return false;
	else{
		m_pQuerry->Add( new TDeviceStruct( tmpDevice, param));
		return true;
	}



}

//////////////////////////////////////////////////////////////////////////////////////////////
// IMPORTANT:
//		No any difference between test command
//		and commands of device.
//		Test coommand catch thread of reding/writing
//		when Cycle or Querry LeaveCriticalSection
//		See code below... ||
//                                                                    \/
bool CSystem::onInteger(DWORD param)
{
 ////////////////////////////////////////////////////////////////////////
 // Firtsful get number of tag if is simple request to execute module
 // specific command, than set this request in querry, if it is request
 // for module test - wait for CriticalSection and test module

	UINT TagNmb = LOBYTE(LOWORD(param));


	CDevice* tmpDevice = IntDevices[TagNmb];

	if (tmpDevice==NULL){  // IMMEDIATLEY TEST MODULE
		tmpDevice = TstDevices[TagNmb];
		if (tmpDevice!=NULL){ //if it is test tag of module test device
			EnterCriticalSection(m_pReadWriteCritical);
```

```
                        tmpDevice->TestModule();
                        LeaveCriticalSection(m_pReadWriteCritical);
                }
                return true;
        }
        else{                   //   SET REQUEST IN QUERRY
                m_pQuerry->Add(new TDeviceStruct(tmpDevice, param));
                return true;
        }

        return false;
}

bool CSystem::onBit(DWORD param)
{
        //1. Tag number from void** param
        UINT TagNmb = LOBYTE(LOWORD(param));

        CDevice* tmpDevice = BitDevices[TagNmb];
        if (tmpDevice==NULL)
                return false;
        else{
                m_pQuerry->Add( new TDeviceStruct(tmpDevice, param));
                return true;
        }
}



void CSystem::m_sysStopThreads()
{
        SuspendThread(m_hCycleThread);
}

void CSystem::m_sysStartThreads()
{
        ResumeThread(m_hCycleThread);
}

//////////////////////////////////////////////////////////////////////////////////////
// IMPORTANT: MCS restart operation cause execution of RESTART command,          //
//        recieving of TableOfResources ('L' command) and verifying it with       //
//                        old table of resources to detect any changes (changes means error   //
//                        situation)
                                                                                //
//
void CSystem::m_sysRestart()
{
        BYTE *NewTableOfResources = new BYTE[255];
        BYTE *Command = new BYTE[6] ;
        ZeroMemory(NewTableOfResources, 255);
        ZeroMemory(Command, 6);
        UINT SzOutput;
        UINT TimeOut = *(m_pFrm->m_tagTimeout);

        Command[2] = 'S';
  //restart MCS
        UINT err = SendCommand( Command, NULL, 6, SzOutput, TimeOut);
        *m_pFrm->m_tagErrorCode = err;

  //get table of resources after restarting MCS
        if (err==OKCODE){
                ZeroMemory(Command, 6);
                Command[2] = 'L';
                *m_pFrm->m_tagErrorCode = err = SendCommand(Command, NewTableOfResources, 6, SzOutput,
TimeOut);
                if (err==OKCODE){
                        *m_pFrm->m_tagValid = MCS_OK; //restart operation success
```

```
                        m_sysVerifyTables(NewTableOfResources);
                }
                else
                        *m_pFrm->m_tagValid = MCS_RESTARTFAILED; //that means the global failure of restart operation
        }
        else{
                *m_pFrm->m_tagValid = MCS_RESTARTFAILED;
        }

        delete [] NewTableOfResources;
        delete [] Command;

};

void CSystem::m_sysVerifyTables(BYTE *pTable)
{
 UINT oldSize =  ( UINT(m_pTableOfResources[3]*256+m_pTableOfResources[2] - 4))/6 - 1; // emount of modules in MCS
 UINT newSize =( UINT(pTable[3]*256+pTable[2] - 4))/6 - 1; // emount of modules in MCS

 if (oldSize!=newSize){
        *m_pFrm->m_tagValid = MCS_CONFIGCHANGED;
 }
 else
        for (UINT Ctr=0; Ctr<oldSize*6; Ctr+=6){
                BYTE oldInd = m_pTableOfResources[4+Ctr];
                BYTE oldNum = m_pTableOfResources[5+Ctr];

                BYTE newInd = pTable[4+Ctr];
                BYTE newNum = pTable[5+Ctr];

                WORD state = MAKEWORD(pTable[8+Ctr], pTable[9+Ctr]);

                if (oldInd==newInd && oldNum==newNum){
                        CDevice* pDevice = FindDevice( newInd, newNum);
                        if (!pDevice){ //that means changes in MCS configuration(Error)
                                *m_pFrm->m_tagValid = MCS_CONFIGCHANGED;
                                return;
                        }
                        else
                                *pDevice->m_tagValid = state;

                }
                else
                        *m_pFrm->m_tagValid = MCS_CONFIGCHANGED;

        } //for (Ctr)

}


CDevice* CSystem::FindDevice(BYTE ind, BYTE num)
{

        CDevice* pDevice = IntDevices.First();
        while (pDevice)
                if (pDevice->m_Index==ind && pDevice->m_NumbFrm==num)
                        return pDevice;
                else
                        pDevice=IntDevices.Next();

        pDevice = AngDevices.First();
        while (pDevice)
                if (pDevice->m_Index==ind && pDevice->m_NumbFrm==num)
                        return pDevice;
                else
                        pDevice = AngDevices.Next();
```

```
	pDevice = BitDevices.First();
	while (pDevice)
		if (pDevice->m_Index==ind && pDevice->m_NumbFrm==num)
			return pDevice;
		else
			pDevice = BitDevices.Next();

	return NULL;
}

void CSystem::mcsStart()
{
	void* param[100];
	param[0] = static_cast<void*>(&IntTag[IntTagCtr++]);
	param[1] = static_cast<void*>(&IntTag[IntTagCtr++]);
	param[2] = static_cast<void*>(&IntTag[IntTagCtr++]);
	param[3] = static_cast<void*>(&BitTag[BitTagCtr++]);
	param[51] = static_cast<void*>(this);
	param[52] = static_cast<void*>(&IntTag[IntTagCtr++]);
	m_pFrm = new CMCSFrame(param);
	IntDevices.Add( m_pFrm, IntTagCtr-2);

 //get resource table from MCS on specified port(m_PortNumber)
	m_IsValid= mcsInitiate();

	if (!m_IsValid){
		*m_pFrm->m_tagValid = MCS_INITFAILED;
		return;
	}
	else
		*m_pFrm->m_tagValid = MCS_OK;

	mcsRegister();
	Started = true;
}

void CSystem::mcsInitAgain()
{
	if (!Started){
		if (!mcsInitiate()){
			*m_pFrm->m_tagValid = MCS_INITFAILED;
		}
		else{
			*m_pFrm->m_tagValid = MCS_OK;
			mcsRegister();
			Started = true;
		};

	}
	else
		return;
}
```

вернуться на “Содержание”

Приложение Г.

1. Потоки обмена данными с МПСУ. Реализация синхронизации.

Класс очереди потока.

```
//class of element of list
template <class CType> class CElement{
 public:
  CElement<CType>* m_pNext;
  CElement<CType>* m_pPrev;
  CType* m_pContent;
  unsigned long m_ulIndex;
 public:
  CElement(){
		m_pContent = NULL;
		m_pNext = NULL;
		m_pPrev=NULL;
  };
  CElement(CType* pContent,unsigned long ulIndex=0){
   m_pContent = pContent;
   m_ulIndex = ulIndex;
   m_pNext = m_pPrev = NULL;
  };
  ~CElement(){
		m_pNext=NULL;
		m_pPrev=NULL;
		if (m_pContent)
			delete m_pContent;
  }
};

template <class CType> class CLinkedList{
protected:
  CElement<CType>* m_pFirst;
  CElement<CType>* m_pLast;
  CElement<CType>* m_pTemp;


  void Add(CType* pContent, unsigned long ulIndex){
		CElement<CType>* New;
		New = new CElement<CType>(pContent, ulIndex);
		ASSERT(New);
		if (!m_pFirst)
			m_pFirst=m_pLast=m_pTemp=New;
		else{
			New->m_pPrev=m_pLast;
			m_pLast->m_pNext=New;
			m_pLast=New;
		}
  };

  void Flush(){
		CElement<CType> *p,*p1;
		p=m_pFirst;
		while (p){
			p1=p->m_pNext;
			delete p;
			p=p1;
		}
  };

  CType* Find(unsigned long ulIndex){
		m_pTemp = m_pFirst;
		while (m_pTemp!=NULL){
			if (m_pTemp->m_ulIndex == ulIndex)
				return m_pTemp->m_pContent;
			m_pTemp = m_pTemp->m_pNext;
```

```
            };
            return NULL;
  };

 public:
  CLinkedList(){ m_pFirst = m_pLast = m_pTemp = NULL;};
  ~CLinkedList(){ Flush();};

  virtual CType* First(){
            if (m_pTemp = m_pFirst)
                        return m_pTemp->m_pContent;
            else
                        return NULL;
  }; //First

  virtual CType* Next(){
            if (m_pTemp = m_pTemp->m_pNext)
                        return m_pTemp->m_pContent;
            else
                        return NULL;
  }; // Next();
};

//////////////////////////////////////////////////////////////////////////////////////////////////
//                    ****************************************                    //
//                    **** Template class of sparse array ****                    //
//                    ***Interface  of  class  CSparseArray***                    //
//                    ****************************************                    //
//////////////////////////////////////////////////////////////////////////////////////////////////
template <class CType>
class CSparseArray : public CLinkedList<CType>
{
 public:
            CType* operator [] (unsigned long ulIndex)
            {
                        return (Find(ulIndex));
            };
            void Add(CType* pContent, unsigned long ulIndex)
            {
                        CLinkedList<CType>::Add(pContent, ulIndex);
            }
            void Flush(){
                        CLinkedList<CType>::Flush();
            }
};
```

вернуться на **“Содержание”**

2. Потоковые функции, реализующие обмен данными с МПСУ

Поток циклического обмена с МПСУ.

```
void CycleThread(LPVOID param)
{
  //pointer to system object
            CSystem* pSystem = static_cast<CSystem*>(param);
            CDevice* pDevice=NULL;
            while (1)
            {
                        pDevice = pSystem->m_pCycle->GetNext();
                        if (pDevice!=NULL){
                                    if ( pDevice->IsInTheThread() )
                                    {
                                                EnterCriticalSection( pSystem->m_pReadWriteCritical);
//!!!IMPORTANT:modules only without //neccessary parameters!!!
                                                pDevice->ExecuteCommand(NULL);
                                                LeaveCriticalSection( pSystem->m_pReadWriteCritical);
```

```
                        }
                };
                Sleep(10);
        }//while(1)

}
```

Поток горячего обмена данными с МПСУ.

```
void QuerryThread(LPVOID param)
{
  //pointer to system object
        CSystem* pSystem = static_cast<CSystem*>(param);
        CSystem::TDeviceStruct* pStruct;
        while (1)
        {
                pStruct = pSystem->m_pQuerry->GetNext();
                if (pStruct!=NULL){
                        EnterCriticalSection( pSystem->m_pReadWriteCritical);
                         (pStruct->pDevice)->ExecuteCommand( pStruct->pParam);
                        LeaveCriticalSection( pSystem->m_pReadWriteCritical);
                        pSystem->m_pQuerry->Remove();
                }
                Sleep(10);
        }//while (1)

}
```

вернуться на **“Содержание”**

Приложение Д.
Описание тегов групп модулей УСО ОРС-сервера МПСУ

1) Описание тегов, входящих в группу модуля дискретного ввода М101

Chanel00…Chanel15 - битовые теги, соответствующие дискретным входам модуля М101. Значения этих тегов отражают состояние дискретных входов модуля М101.

ReadCode – битовый тег, запись значения On в который приводит к немедленному чтению состояния каналов модуля дискретного ввода М101.

2) Описание тегов, входящих в группу модуля дискретного вывода М102

Chanel00…Chanel15 - битовые теги, соответствующие дискретным выходам модуля М102. Значения этих тегов отражают состояние дискретных выходов модуля М102. При условии нахождения данной группы в потоке циклического обмена с МПСУ (тег IsInThread имеет значение On) запись значения в один из этих тегов влечет за собой пересылку нового состояния выходов в МПСУ.

WriteCode – битовый тег, запись значения On в который приводит к немедленной записи нового состояния каналов модуля дискретного вывода М102 в МПСУ.

3) Описание тегов, входящих в группу модуля дискретного вывода М103

Аналогично группе модуля дискретного вывода М102.

4) Описание тегов, входящих в группу модуля аналогового ввода М113

Chanel00…Chanel07 - вещественные теги, соответствующие аналоговым входам модуля М113. Значения этих тегов отражают состояние аналоговых входов модуля М113.

NumberOfChanel – целочисленный тег, значение которого содержит номер читаемого аналогового входа модуля М113. При записи нового значения в этот тег производится моментальное чтение вновь указанного канала модуля. По завершении операции записи значение тега сбрасывается в Off.

5) Описание тегов, входящих в группу модуля дискретного ввода М201

Chanel1_00…Chanel1_15 – битовые теги, значение которых отражает состояние дискретных входов младшего регистра модуля М201.

Chanel2_00…Chanel2_15 – битовые теги, значение которых отражает состояние дискретных входов старшего регистра модуля М201.

ReadCode - битовый тег, запись значения On в который приводит к немедленному чтению состояния каналов модуля дискретного ввода М201. По завершении операции чтения значение тега сбрасывается в Off.

6) Описание тегов, входящих в группу модуля дискретного вывода М202

Chanel00…Chanel08 - битовые теги, соответствующие дискретным выходам модуля М202. Значения этих тегов отражают состояние дискретных выходов модуля М202. При условии нахождения данной группы в потоке циклического обмена с МПСУ(тег IsInThread имеет значение On) запись значения в один из этих тегов влечет за собой пересылку нового состояния выходов в МПСУ.

WriteCode – битовый тег, запись значения On в который приводит к немедленной записи нового состояния каналов модуля дискретного вывода М202 в МПСУ. По завершении операции записи значение тега сбрасывается в Off.

7) Описание тегов, входящих в группу модуля дискретного вывода М203

Chanel1_00…Chanel1_15 – битовые теги, значение которых отражает состояние дискретных входов младшего регистра модуля М203.

Chanel2_00…Chanel2_15 – битовые теги, значение которых отражает состояние дискретных входов старшего регистра модуля М203.

При условии нахождения данной группы в потоке циклического обмена с МПСУ(тег IsInThread имеет значение On) запись значения в один из этих тегов влечет за собой пересылку нового состояния выходов в МПСУ.

NumberOfRegister – целочисленный тег, сообщающий ОРС серверу о том, в какой из регистров производить запись значений.

Значение 0 – младший регистр, значение 1-старший.

WriteCode - битовый тег, запись значения On в который приводит к немедленной записи состояния каналов выбранного в данный момент регистра модуля дискретного вывода М203. По завершении операции чтения значение тега сбрасывается в Off.

8) Описание тегов, входящих в группу модуля аналогового ввода М204

Chanel01…Chanel08 –вещественные теги, соответствующих каналов модуля аналогового ввода М204. Значение этих тегов содержат значения аналоговых входов модуля М204.

Mask – целочисленный тег, содержащий значение маски читаемых каналов, представленный в десятичной форме. Запись нового значения ОРС клиентом в этот тег приводит к немедленному чтению состояния аналоговых входов модуля М204 в соответствии с новой маской (модуль помещается в поток «горячего» обмена данными с МПСУ).

9) Описание тегов, входящих в группу модуля аналогового вывода М210

Chanel01…Chanel15 –вещественные теги, соответствующих каналов модуля аналогового вывода М210. Запись нового значения в один из этих тегов влечет за собой запись в соответствующий канал модуля М210.

10) Описание тегов, входящих в группу модуля 32-х 16 битных счетчиков М230

Counter00 … Counter32 – целочисленные теги, значение тегов соответствует последним прочитанным значениям счетчиков модуля М230 или последним записанным начальным значениям счетчиков модуля М230. Запись нового значения в указанные теги влечет за собой немедленную запись начального значения соответствующего счетчика модуля М230.

Command – битовый тег, запись значения On в который влечет за собой немедленное чтение текущего состояния 32 счетчиков модуля М230. Прочитанные значения счетчиков записываются в теги **Counter00 … Counter32**

11) Описание тегов, входящих в группу дополнительного модуля пользователя MY01

OutputData – целочисленный тег, запись значения в который, влечет за собой немедленную запись этого же значения в буфер модуля MY01.

InputData - целочисленный тег, содержащий последнее прочитанное значение, содержащееся в буфере модуля MY01.

ReadData – битовый тег, запись значения On в который влечет за собой немедленное чтение содержимого буфера модуля MY01, по завершении операции чтения значение тега сбрасывается в Off.

вернуться на **“Содержание”**