Распределенная информационная система управления DIM

Distributed Information Management System

Аннотация

DIM - это коммуникационная программная технология для создания распределенных систем управления в неоднородных (смешанных) средах, работающих на разных аппаратных и программных платформах. Она предоставляет переносимый, прозрачный для сети механизм взаимодействия между процессами, работающими на различных компьютерах в сети. Это руководство предоставляет обзорное описание функционала **DIM** и его использования.

С точки зрения прикладного программиста, **DIM** представляет собой динамическую (**DIM.DLL**) или статическую (**DIM.LIB**) библиотеку с интерфейсами для разных языков программирования (**C**, **C++**, **Fortran**, **Java**, **Pascal**, **LabView**), а также набор служебных утилит (**DNS.EXE**, **DID**, **DIMTREE**, **DIMSTAT**), распространяемых по лицензии **GPL**.

Система **DIM** написана на **C/C++**, доступна в исходных кодах, работает под **Windows**, **Linux**, **Unix**. При необходимости она может быть перенесена и на другие платформы. На сетевом уровне **DIM** базируется на сокетах **TCP/IP**, используя (при стандартных настройках) порты **TCP** 2505,5100,5101,5102,... (число используемых портов зависит от числа работающих на компьютере **DIM** серверов).

Система **DIM** доступна на сайте http://dim.web.cern.ch

Оглавление

Введение....3
1 История и примеры использования DIM....4
2 Принципы функционирования DIM....6
2.1 Основы DIM: сервисы, серверы, клиенты....6
2.2 Сервер имен DNS....7
2.3 Асинхронный режим DIM....8
2.4 Доступность и инструменты....10
2.4.1 Утилита DID....10
2.4.2 Утилита DIMTREE....11
2.4.3 Утилита DIMSTAT....12
2.5 Соглашения по наименованию....13
2.6 Соглашения по описанию типов....15
2.7 Сетевая адресация DIM....16
2.8 Потоковая модель DIM....16
2.9 Временные параметры DIM....16
2.10 Производительность DIM....17
2.11 Множественная конфигурация DIM....18
2.12 Заметки и Примечания....19
2.13 Недостатки DIM....19
2.14 Некоторые рекомендации по проектированию....20
3 Практические вопросы использования DIM....21
3.1 Необходимые файлы времени исполнения....21
3.2 Переменные окружения....22
3.3 Используемые DIM порты TCP/IP....23
3.4 Сетевые службы DIM....23
3.5 Сетевой мост DIM Bridge....24
4 Программный интерфейс DIM API....25
4.1 Обзор состава библиотек DIM....25
4.2 Обзор программного интерфейса DIM API....27
4.3 Структура типичного DIM сервера....28
4.4 Структура типичного DIM клиента....29
4.5 Библиотеки классов для DIM....30
4.6 Пользовательские процедуры (callback)....30
Заключение....32
Список использованных источников....33
Приложение 1. Состав архива DIM....35
Приложение 2. Модуль _DIM.PAS....40
Приложение 3. Программа SERVER.DPR....49
Приложение 4. Программа CLIENT.DPR....53
Приложение 5. Скрипт DEMO.BAT....56
Приложение 6. Скрипт DIMPORTS.CMD....57
Приложение 7. Шаблон спецификации DIM сервера....59

Введение

DIM [1,2] - сокращение от «Distributed Information Management System» – «распределенная система управления информацией». Она разработана в **CERN** [3] (Европейский центр ядерных исследований, Женева) и используется для управления рядом крупных экспериментов на Большом Адронном Коллайдере (**LHC**).

Система **DIM** прекрасно масштабируется. Она разработана специально для больших экспериментов в области физики высоких энергий, в которых совместно работают сотни компьютеров, фактически работая как один суперкомпьютер. В то же время **DIM** – довольно компактная и высокоэффективная технология, которая прекрасно работает и в маленьких системах, где число серверов и клиентов невелико.

В настоящее время на базе **DIM** работают практически все крупные эксперименты в **CERN**. Управление самим ускорителем **LHC** (а это самый крупный в мире ускорительный комплекс) также сделано на основе **DIM**. В области физики высоких энергий и ускорителей **DIM** весьма популярен и стал фактическим стандартом.

Большим достоинством **DIM** является её доступность в исходных кодах, свободная лицензия (**GPL**), высокая переносимость и поддержка большого числа аппаратных платформ, операционных систем и языков программирования. Это позволяет использовать **DIM** в смешанных средах, состоящих из компьютеров, работающих на самых разных аппаратных и программных платформах. Эти свойства, а также свободная лицензия делают возможным использование **DIM** в любых проектах, в том числе за пределами **CERN**.

Наконец, удобные универсальные утилиты для наблюдения и отладки клиентов и серверов делают разработку распределенных систем с помощью **DIM** удобной и высокоэффективной.

Начиная с 2005 года система **DIM** была интегрирована в пакет **CRW-DAQ** [4,5], разработанный в **ИЯРФ** для автоматизации измерительных систем и исследовательских физических установок. В составе этого пакета система **DIM** более 12 лет успешно работает на множестве исследовательских установок в Сарове (**ВНИИЭФ**), Дубне (**ОИЯИ**), Женеве (**CERN**). Накопленный опыт подтвердил высокую надежность и отказоустойчивость системы **DIM**, её эффективность и удобство в работе.

В данном документе описываются основные свойства системы **DIM**, принципы её функционирования, а также даются начальные сведения о программировании клиентов и серверов в системе **DIM**.

1 История и примеры использования DIM

Технология **DIM** создавалась начиная с (примерно) 1993 года в рамках эксперимента **Delphi** на ускорителе **LEP** (*Large Electron-Positron collider*) в **CERN** [2]. Система управления эксперимента **Delphi** включала в себя более 500 процессов, работающих на более 50 серверах, работающих в основном под управлением **OS VMS** и **OS9**. В процессе развития программный код **DIM** был адаптирован и внедрен на множестве других программных и аппаратных платформ (**Unix**, **Linux**, **Windows** и т. д.) и использовался в ряде других экспериментов **CERN** (**NA50**, **L3**, **BaBar**) [2].

Технология **DIM** хорошо апробирована, испытана и уже много (более 15) лет используется в крупных экспериментах, проводимых в **CERN**. Здесь приведены только некоторые примеры использования **DIM** на установках **CERN**, перечислить все было бы затруднительно из-за большого их количества.

Технология **DIM** используется в системе управления детектора (**DCS**, *Detector Control System*) эксперимента **ALICE** [7] на Большом Адронном Коллайдере (**БАК** или **LHC**, *Large Hadron Collider*) в **CERN**, которая включает в себя более 100 серверов и контролирует около 1000000 параметров [8]. В частности, **DIM** используется в системах контроля и управления детектора **TRD** [9], стартового детектора **T0** и времяпролетной системы **ALICE** [10], электромагнитного спектрометра **PHOS** [11] и в других 19 детекторах, входящих в состав **ALICE**. На базе протокола **DIM** работает распределенная Машина Состояний (**SMI**, *State Machine Interface*) [12,13,14], реализующая многоуровневый Конечный Автомат (**FSM**, *Finite State Machine*), с помощью которого ведется управление экспериментом.

Технология **DIM** используется как стандартная много-платформенная среда передачи для взаимодействия процессов (*CERN standard multi-platform middle-ware for inter-process communication*) в системе управления детектора (**DCS**) эксперимента **ATLAS** [15,16] на Большом Адронном Коллайдере (**LHC**) в **CERN**, которая включает около 200000 датчиков. Управление экспериментом **ATLAS** также выполняется с помощью Машины Состояний (**SMI**) на базе **DIM**. Компьютеры в составе эксперимента работают под различными операционными системами (в основном **Linux** и **Windows**). Это делает **DIM** особенно важным связующим компонентом системы, т. к. некоторые технологии (например, **OPC**) не являются много-платформенными и работают только под **Windows**.

Технология **DIM** используется как одна из основных сетевых технологий среднего уровня (*middleware layer*) для связи между подсистемами управления разного уровня в эксперименте **LHCb** [17]. Для построения распределенной системы управления экспериментом **LHCb** также используется распределенная Машина Состояний (**SMI** – *State Machine Interface*), работающая на основе протокола **DIM**.

На базе **DIM** как одного из основных протоколов связи работает система управления экспериментом **NA62** [18] по изучению распада каонов на ускорителе **SPS** в **CERN**. Эта система имеет около 11000 каналов регистрации, включает разнообразное оборудование (как промышленное, так и специально разработанное) и серверы, работающие под **Linux** или **Windows**.

Технология **DIM** служит базовым протоколом, на основе которого строятся другие (вторичные) протоколы связи с дополнительными функциями. Например, кроме упоминавшейся выше Машины Состояний (**SMI**) на основе **DIM** реализован протокол **DIP** (*Data Interchange Protocol*) [19], на основе которого построена система управления ускорителя **LHC**, а также с помощью которого осуществляется связь между службами ускорителя (**LHC**) и детекторами (**ATLAS**, **CMS**, **ALICE**, **LHCb**). Вторичные протоколы связи добавляют к базовым функциям **DIM** дополнительные соглашения, библиотеки и инструменты, реализующие требуемые дополнительные возможности.

Для поддержки и унификации протоколов связи и средств разработки систем управления в **CERN** была создана специальная группа **JCOP** (*Joint COntrols Project*). В её задачи (в частности) входит поддержка **DIM**, **DIP**, **SMI** и других протоколов, используемых на установках **CERN**.

С использованием протоколов **DIM** и **DIP** построена система управления, а также удаленного **Web** мониторинга для эксперимента **CMS** [21]. Система мониторинга использует **Web** технологии (на основе протокола **HTTP**) для организации интерфейса удаленного пользователя, в то время как **DIM** и **DIP** используется для связи этого интерфейса с системами управления эксперимента **CMS**. Таким образом, технология **DIM** и вторичные протоколы на её основе не мешают использованию других протоколов связи, а напротив, гармонично сочетаются с ними.

Установки с использованием **DIM** создаются и за пределами **CERN**. Например, протокол **DIM** использовался в системе регистрации для изучения спектров космических излучений на детекторе **L3** [22].

На базе технологии **DIM** построена система управления эксперимента **CLOUD** в **CERN** по изучению космических частиц [23].

В целом можно сделать вывод, что **DIM** является зрелой, хорошо развитой и проверенной на большом числе крупных установок технологией для организации распределенных много-платформенных систем управления. Эта технология хорошо масштабируется и может применяться как для небольших (несколько серверов), так и крупных (сотни серверов) систем.

2 Принципы функционирования DIM

DIM, как большинство коммуникационных технологий, основана на парадигме клиент/сервер. Под сервером здесь понимается процесс, владеющий некоторыми данными, которые он может публиковать, т. е. предоставлять другим (клиентским) процессам в сети. Клиентами являются различные процессы - потребители данных, расположенные как локально, так и на любых других компьютерах в сети. В типичной ситуации сервером является процесс, управляющий измерительной аппаратурой, а клиентами — любые процессы, которые используют измеренные данные.

2.1 Основы DIM: сервисы, серверы, клиенты

Базовой концепцией **DIM** является понятие «**сервис**» (service). Серверы предоставляют клиентам сервисы. Сервис — это некоторый набор данных (определенного типа и размера), который распознается по имени - «**именованный сервис**» (named service). Пространство имен сервисов в целом произвольно, но подчиняется ряду простых правил (см. раздел 2.5). Сервисы также делятся по назначению на класс **информационных** (для передачи данных от сервера клиентам) и **командных** (для приема сервером команд от клиентов). Кроме имени, у сервиса есть также (необязательный) **описатель типа**, указывающий на типы хранимых в сервисе данных (см. раздел 2.6). Сервисы рассматриваются как двоичные данные, т. е. **DIM** никак не интерпретирует содержимое сервисов и не накладывает на это содержимое никаких существенных ограничений.

Обычно клиенты запрашивают сервисы только один раз (при старте), а затем они автоматически получают обновления данных от сервера — в зависимости от настройки - либо по таймеру, либо по мере надобности (т. е. при появлении новых данных у сервера). Механизм обновления клиента бывает двух типов — путем выполнения пользовательской процедуры (**callback routine**) или путем обновления клиентского буфера, или оба сразу. По факту это работает так, как будто у клиента есть кэш с копией данных сервера, причем когерентность данных поддерживает сам **DIM**, без специальных усилий со стороны клиента.

Для обеспечения сетевой прозрачности (когда клиенту не требуется знать, где работает сервер), а также для облегчения восстановления после разрыва связи или миграции сервера, в технологии **DIM** используется «**сервер имен**» - **DNS** (dynamic name server).

Серверы «**публикуют**» (publish) свои сервисы, регистрируя их на сервере имен (обычно один раз, при старте). При изменении данных серверы автоматически обновляют сервисы для всех подключенных к ним клиентов.

Клиенты «**подписываются**» (subscribe) на нужные им сервисы, запрашивая у сервера имен информацию о том, какой сервер предоставляет интересующий сервис, и затем связываются с этим сервером напрямую, запрашивая у него обслуживание требуемого типа (по таймеру или с автоматическим обновлением). Для изменения состояния сервера клиенты посылают ему **команды**, которые сервер принимает к исполнению.

2.2 Сервер имен DNS

Сервер имен **DNS** (см. Примечания 2.12) хранит актуальный каталог всех работающих **DIM** серверов и принадлежащих им сервисов, доступных в **DIM** системе. Он обеспечивает сетевую прозрачность и высокую отказоустойчивость связи. При сбое **DIM** сервера или в случае его миграции на другой компьютер связь восстанавливается автоматически, т. к. при запуске **DIM** сервера он немедленно выполняет регистрацию сервисов на сервере имен и сразу становится доступным для всех клиентов.

Следующий рисунок 1 показывает, как взаимодействуют компоненты **DIM** (Серверы, Клиенты и Сервер Имен).

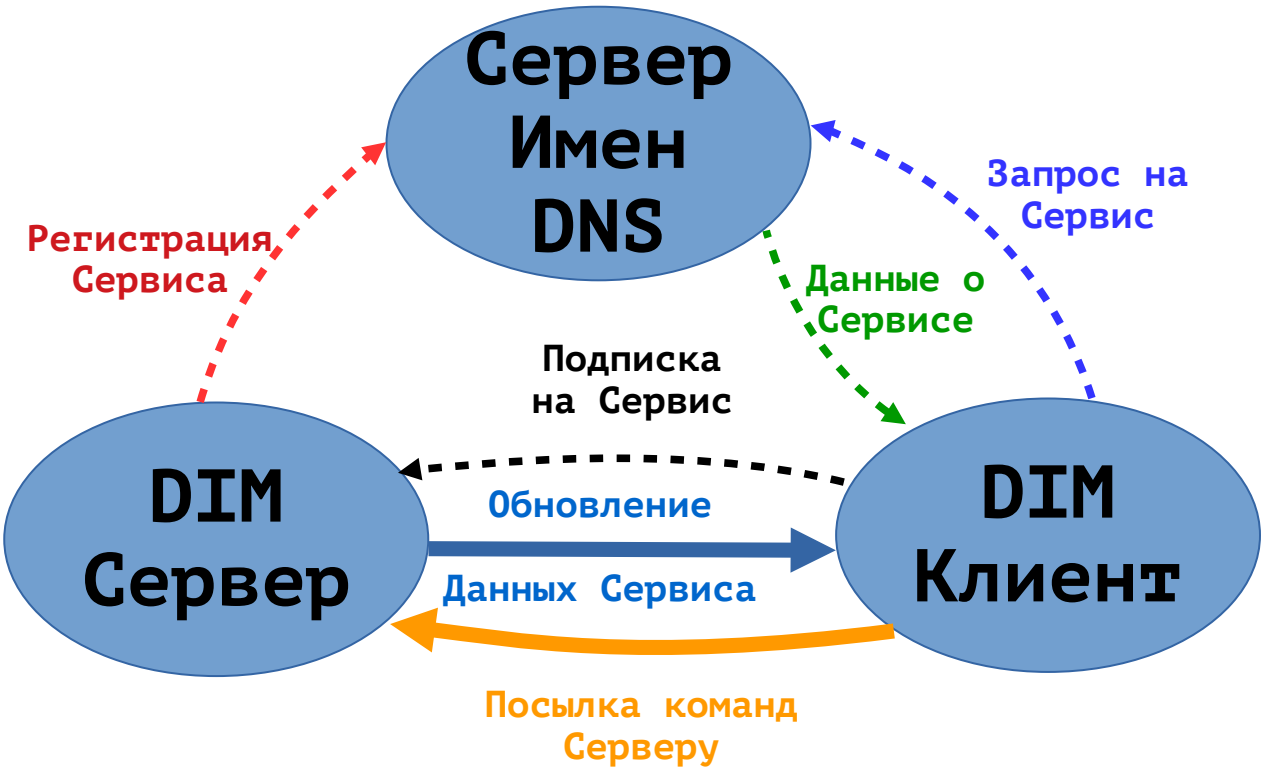


Рисунок 1 – схема взаимодействия **DIM** сервера, **DIM** клиента и сервера имен **DNS**.

В случае сбоя или аварийного завершения одного из процессов (**DIM** сервера или даже сервера имен **DNS**) все подключенные к ним процессы **DIM** клиентов оповещаются и автоматически восстанавливают подключение, как только серверный процесс возвращается к жизни. Это свойство не только позволяет облегчить восстановление после сбоев, но и дает возможность серверному процессу мигрировать с одной машины на другую (путем остановки сервера на одной машине и запуска на другой), позволяя балансировать нагрузку на разных рабочих станциях или осуществлять «горячее резервирование» ответственных подсистем.

В целом можно сказать, что **DIM** берет на себя всю «черную работу» по обеспечению связи между процессами и «скрывает» от пользователя сложность сетевого взаимодействия, позволяя программисту целиком сосредоточиться на решении основных задач (получении и обработке данных), мало заботясь о том, как именно данные передаются между сервером и клиентом.

2.3 Асинхронный режим DIM

В отличие от классической синхронной модели взаимодействия, когда клиент посылает серверу запрос и ждет от него ответа, технология **DIM**, подобно **OPC** [6] использует асинхронную модель взаимодействия. Она состоит в том, что клиенты «подписываются» на интересующий сервис и продолжают свою работу, не ожидая ответа сервера, а сервер по таймеру или при изменении данных сервиса делает «обновление», т. е. рассылку данных сервиса всем подписанным на этот сервис клиентам (Рис.2). Асинхронная модель позволяет избежать циклов ожидания, что очень важно для задач управления. Клиенты **DIM** также могут асинхронно посылать серверу команды, позволяющие управлять его состоянием.

Есть две диаметрально противоположные парадигмы организации систем клиент-сервер (Рис.2). Первая парадигма (Рис.2,а), именуемая часто **RPC** (Remote Procedure Call) предполагает, что сервер - владелец информационных ресурсов, обслуживает запросы клиентов, являющихся потребителями этих ресурсов, которые берут у сервера интересующие данные и отсылают ему результаты обработки, если это нужно. Взаимодействие в **RPC** происходит по принципу «Запрос-Ожидание-Ответ» (Request-Waiting-Reply), причем активной стороной является клиент, посылающий запросы. Большинство систем управления базами данных (СУБД) и Web технологий основано на этой парадигме (например, **SQL** и **HTTP**).

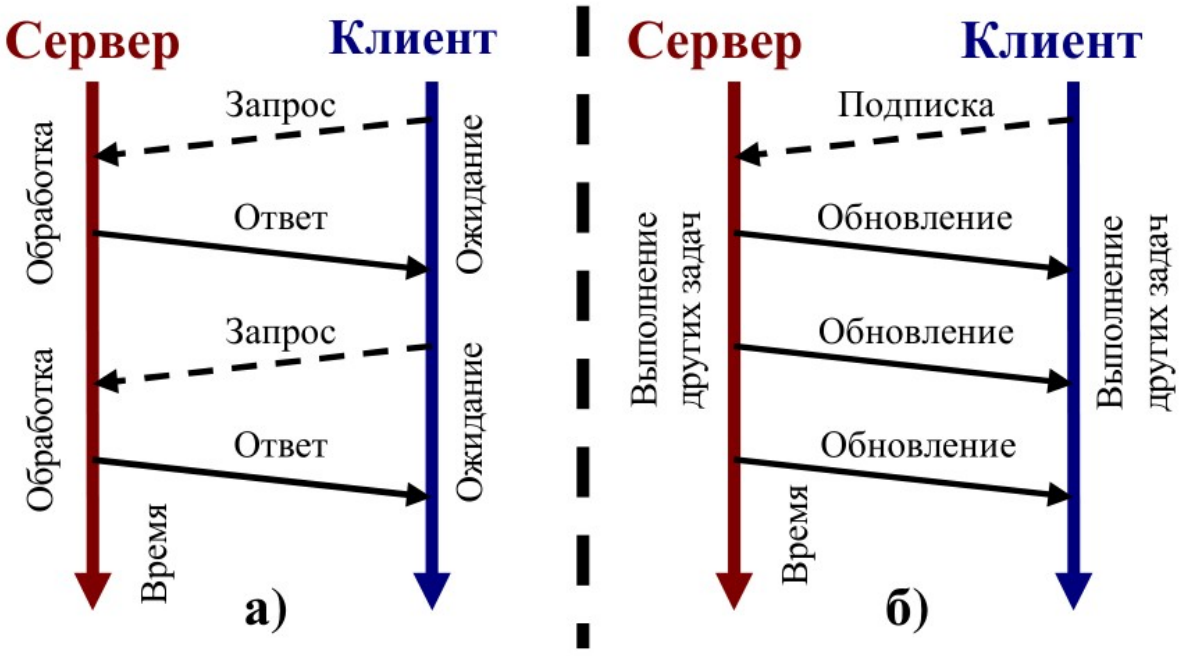


Рисунок 2 – Сравнение двух парадигм взаимодействия клиент-сервер:
а) запрос-ожидание-ответ (HTTP, SQL…),
б) подписка-обновление (DIM, OPC…).

Вторая парадигма (Рис.2,б), реализованная, например, в сетевых технологиях **DIM** и **OPC**, основана на принципе «Подписка-Обновление» (Subscribe-Refresh). При этом клиент однократно (при подключении к серверу) выполняет подписку (Subscribe) на интересующий

информационный сервис (Service), которым владеет сервер (например, на данные измерительной системы), и продолжает решение клиентских задач без ожидания ответа от сервера. При поступлении новых данных сервер автоматически рассылает обновления (Refresh) всем клиентам, которые подписались на обновляемые информационные ресурсы. Активной стороной при этом является сервер, а обновление данных на стороне клиента имеет принудительный и асинхронный характер, обычно реализуемый через обратный процедурный вызов (Callback). Использование принципа «Подписка-Обновление» характерно для систем управления реального времени.

Сравнивая две парадигмы взаимодействия клиент-сервер, надо в первую очередь понимать, что они ставят разные цели, и поэтому каждая из них оптимальна - для своего круга задач. Принцип **RPC** минимизирует трафик между клиентом и сервером. Он оптимален для большинства СУБД, когда клиента интересует только малая часть из большого серверного хранилища данных. При этом клиент посылает запрос, описывая требуемый информационный ресурс (например, URL в случае **HTTP** или **SQL** запрос в случае СУБД). Сервер делает выборку и отсылает клиенту только то, что он запросил. В то же время в **RPC** всегда присутствует неопределенное по времени ожидание ответа на запрос. Кроме того, активной стороной **RPC** является клиент, поэтому если клиент по каким-то причинам не отправляет серверу запрос, он не будет вовремя уведомлен о состоянии объекта, даже если с ним случилась авария. Это недопустимо для систем управления, где извещение оператора о возникшей аварии должно быть незамедлительным и принудительным.

Принцип «Подписка-Обновление» ставит целью не минимальный трафик, а минимальное время отклика системы на события в объекте управления. Если клиент подписан на информационный ресурс, он немедленно и принудительно получит его обновление, даже если в данный момент это не представляет для него интерес, поэтому трафик заведомо не будет оптимальным по объему. Однако в данной парадигме отсутствует ожидание, как на стороне сервера, так и на стороне клиентов, которые оповещаются о событиях в системе по инициативе сервера немедленно и принудительно. Для систем управления потенциально опасными объектами в реальном времени требования к своевременному оповещению и времени реакции системы важнее минимизации сетевого трафика, обычно небольшого. Поэтому принцип «Подписка-Обновление» лежит в основе **DIM**, **OPC** и большинства систем **SCADA** (Supervisory Control and Data Acquisition).

Таким образом, асинхронный режим работы системы **DIM** делает её подходящей технологической базой для создания систем управления (мягкого) реального времени, контрольно-измерительных комплексов, **SCADA** систем и т. д.

Отметим, что в пакете **CRW-DAQ** [4,5] система **DIM** была выбрана в качестве основной сетевой технологии, используемой для создания распределенных систем управления.

2.4 Доступность и инструменты

Система **DIM** доступна в смешанной среде для различных аппаратных платформ и операционных систем, включая: **VMS**, **Unix**, **Linux**, **Windows** и системы реального времени: **OS9**, **LynxOs** и **VxWorks**. Она использует сетевой протокол **TCP/IP**. Обычно **DNS** использует порт **TCP**:2505, а серверы – **TCP**:5100,5101,5102,… (каждый сервер занимает 3 порта).

Различия в представлении данных (порядок следования байтов, форматы вещественных чисел, выравнивание данных и разница в размерах чисел) для машин с разной архитектурой автоматически (прозрачно для пользователей) согласуется между сервером, клиентом и сервером имен.

Вся функциональность **DIM** доступна для клиентских и серверных приложений в виде статических или динамических библиотек (**DLL, LIB**, **SO**), предоставляющих программные интерфейсы для популярных языков программирования (**C**, **C++**, **Fortran, Java**, **Pascal, LabView**).

Поведение сложных распределенных систем может быть очень сложным для понимания, если нет подходящих **инструментов** для наблюдения и отладки. **DIM** система предоставляет такие инструменты - утилиты, позволяющие наблюдать и отлаживать процессы взаимодействия серверов и клиентов в реальном времени.

2.4.1 Утилита DID

Утилита **DID** (Distributed Information Display), внешний вид которой показан на рисунке 3, доступна для всех платформ.

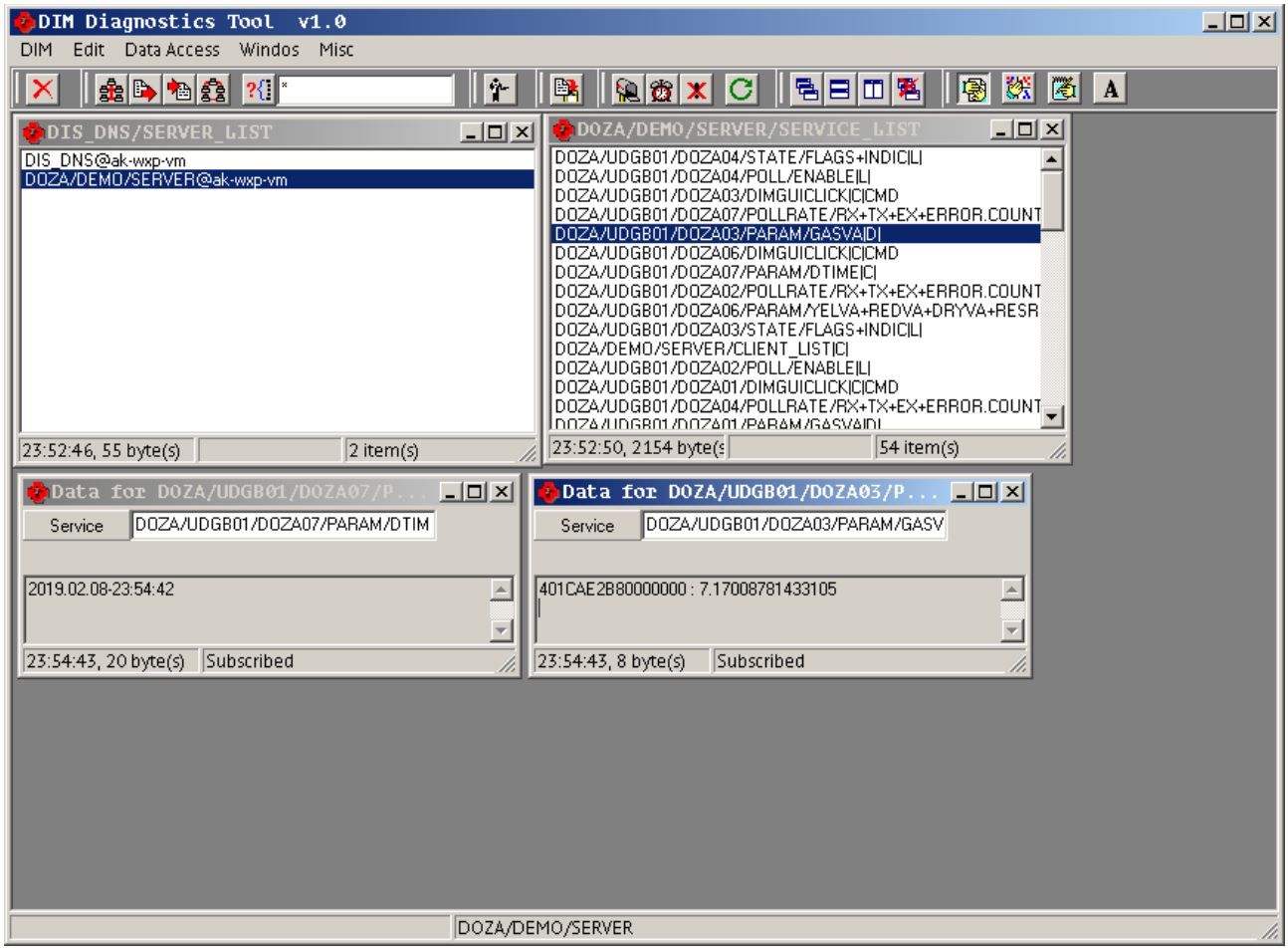


Рисунок 3 – внешний вид программы **DID**.

Утилита **DID** позволяет в реальном времени наблюдать список серверов, подключенных к серверу имен **DNS**, для каждого сервера – список его сервисов и подключенных клиентов, для каждого сервиса – его содержимое. На каждый сервис можно подписаться, тогда он будет обновляться автоматически.

При использовании утилиты **DID** следует помнить, что она требует для работы имени **DNS** сервера, которое можно задать двумя способами – через переменную окружения **DIM_DNS_NODE** или через аргумент:

Способ 1 – переменная окружения	Способ 2 – аргумент вызова
set DIM_DNS_NODE=%ComputerName% **did.exe**	**did.exe %ComputerName%**

2.4.2 Утилита DIMTREE

Утилита **DIMTREE** (DIM Tree Browser), внешний которой показан на рисунке 4, доступна только под **Windows**. Она позволяет в реальном времени наблюдать список серверов, подключенных к данному серверу имен **DNS**, а также все публикуемые этими серверами сервисы вместе с их содержимым в виде многоуровневого дерева, например:

- Имя Сервера — DOZA/DEMO/SERVER
  - Имя сервиса — DOZA/UDGB01/DOZA01/PARAM/DTIME
    - Данные сервиса — 2019.02.08-23:31:45

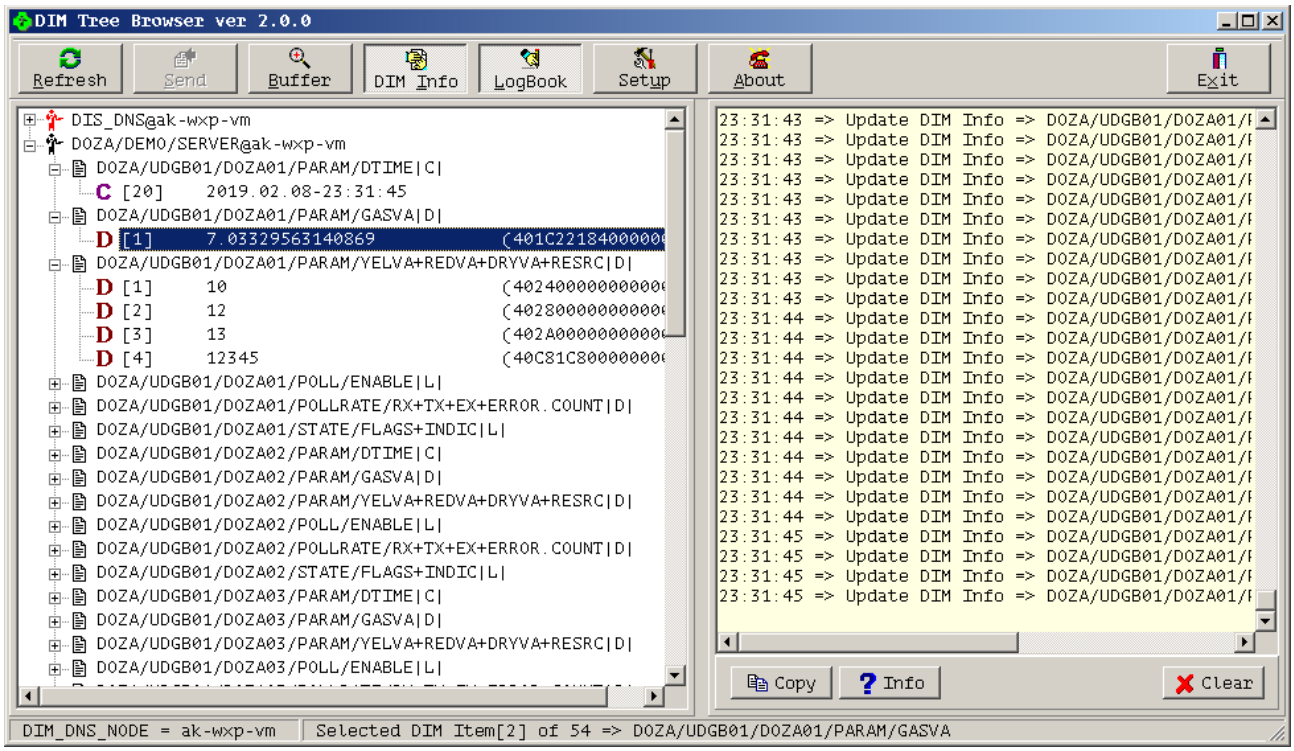


Рисунок 4 – внешний вид программы **DIMTREE**.

Утилита **DIMTREE** является универсальным инструментом отладки. Она позволяет наблюдать публикуемые **DIM** сервером данные, не создавая для этого специального клиента. Если данные видны в окне **DIMTREE**, то всё будет работать и в рабочем варианте клиента. Поэтому создатели **DIM** сервера могут отлаживать свою часть **ПО** независимо от клиентов – потребителей публикуемых данных.

2.4.3 Утилита DIMSTAT

Консольная утилита **DIMSTAT.EXE** (от слов **DIM Statistics**) является мощным инструментом для анализа статистики и сетевого трафика **DIM**.

Утилита **DIMSTAT** также является универсальным инструментом, пригодным для диагностики и отладки любых сетей **DIM**. Она позволяет проверять доступность сервера имен **DIM DNS**, получать список работающих серверов, список подключенных к ним клиентов, список публикуемых ими сервисов. Также она позволяет подключаться к заданным сервисам (по маске имени) и измерять частоту обновлений сервисов (**Poll Rate**, число опросов в секунду) и частоту передачи данных (**Byte Rate**, число байт в секунду). Это позволяет анализировать трафик, генерируемый **DIM** серверами и определять потенциальные «слабые места» или ошибки в работе **DIM** серверов.

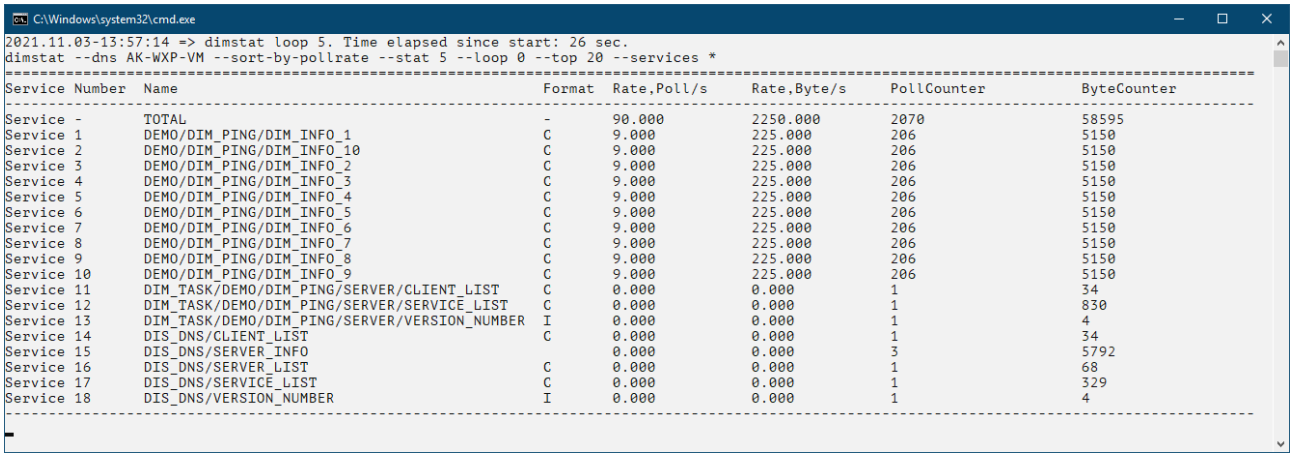


Рисунок 5 – внешний вид программы **DIMSTAT**.

Утилита **DIMSTAT** имеет большое число опций, для справки вызывается команда `dimstat --help`. В качестве примера на рисунке 5 показан результат вывода команды:

```
dimstat --dns AK-WXP-VM --sort-by-pollrate --stat 5 --loop 0 --top 20 --services *
```

Эта команда подключается к заданному серверу имен **DIM DNS** (опция `--dns AK-WXP-VM`), подключается ко всем его сервисам (опция `--service *`), и в бесконечном цикле (опция `--loop 0`) набирает статистику обновлений за 5 секунд (опция `--stat 5`), отображая первые 20 пунктов таблицы (опция `--top 20`), отсортированной по частоте обновлений (опция `--sort-by-pollrate`). В результате можно в реальном времени наблюдать частоту обновлений и трафик работающих серверов и публикуемых сервисов.

В состав сборки **DIM** и пакета **CRW-DAQ** входят и другие отладочные утилиты, например, тестовые серверы и клиенты для проверки и отладки работы **DIM**. Все они доступны в исходных кодах и могут служить прототипом для написания рабочих серверов и клиентов.

Наличие универсальных утилит для наблюдения серверов и сервисов является важным аргументом в пользу **DIM**, т. к. наличие таких инструментов позволяет полностью разделить задачи создания серверов и клиентов. При написании **DIM** сервера нет необходимости заботиться о клиентском **ПО**, т. к. всю отладку можно вести с помощью **DIMTREE** и **DID**. При этом клиенты тоже могут разрабатывать свои программы независимо, им достаточно знать имена публикуемых сервисов. Это сильно облегчает процесс разработки прикладного **ПО**.

2.5 Соглашения по наименованию

DIM идентифицирует **DIM** серверы и публикуемые ими сервисы по именам. Выбор имен является важным вопросом, т. к. в одной сети не допустим конфликт имен разных сервисов. Другими словами, имена серверов и сервисов должны быть уникальными в пределах одной сети.

Как было сказано выше, именование серверов и сервисов в целом произвольно, но надо соблюдать ряд простых правил выбора имен:

- Не использовать в именах специальные (непечатные) символы и пробелы, символы пунктуации, а также символы **«|»**, **«@»**,которые используется как разделители.
- Имена не должны начинаться с символов **«+»**, **«-»**, **«!»**, которые имеют для **DIM** специальное значение. В других местах имени (не в начале) использовать эти символы допустимо.
- Имена серверов и сервисов не должны совпадать с постоянными именами служебных серверов и сервисов, используемых **DIM** для внутренних целей, которые перечислены ниже: сервер **DIS_DNS**, (это имя сервера имен); сервисы: **SERVER_LIST**, **CLIENT_LIST**, **SERVICE_LIST**, **SET_EXIT_HANDLER**, **EXIT**, **VERSION_NUMBER**).

Несмотря на то, что других ограничений на имена в общем-то нет, рекомендуется следовать устоявшейся практике именования сервисов:

- Использовать имена, состоящие из слов, разделенных символом **«/»** (слэш), что делает их похожими на пути файлов **Unix**.
- Использовать в именах сервисов верхний регистр символов, чтобы заранее избежать возможных ошибок из-за неверного регистра (т. к. имена сервисов чувствительны к регистру).
- Использовать единый префикс для всех имен данного **DIM** сервера. Это позволит избежать конфликта имен сервисов от разных серверов, которые будут отличаться префиксами имен.
- Использовать для имен серверов префикс **DIM_TASK**.

Например, если у нас есть сервер для установки **DOZA**, то хорошими именами для **DIM** сервера и его сервисов будут примерно такие:

- **DIM_TASK/DOZA/DEMO/SERVER** – имя сервера, префикс **DIM_TASK**
- **DOZA/UDGB01/DOZA01/PAR/DTIME** – имя сервиса, параметр **DTIME**
- **DOZA/UDGB01/DOZA01/PAR/GASVA** – имя сервиса, параметр **GASVA**

В этом примере название установки (**DOZA**) служит общим префиксом, компоненты имени кратко описывают назначение сервиса (например, **UDGB01** – тип прибора, **DOZA01** – номер прибора, **PAR** – указывает на измеряемый параметр, **DTIME**, **GASVA** – имя параметра).

Следует заметить, что **DIM** хранит имена серверов и сервисов в виде сервисов определенного формата, что можно наблюдать утилитами **DID**, **DIMTREE**. Каждый сервер (**SERVERNAME**) имеет, кроме сервисов пользователя, ряд служебных сервисов с постоянными именами:

- **SERVERNAME/SERVICE_LIST** – список публикуемых сервисов
- **SERVERNAME/CLIENT_LIST** – список подключенных клиентов
- **SERVERNAME/EXIT** – команда завершения сервера
- **SERVERNAME/SET_EXIT_HANDLER** – установка процедуры завершения
- **SERVERNAME/VERSION_NUMBER** – номер версии **DIM** сервера

Список сервисов **SERVERNAME/SERVICE_LIST** хранится в виде сервиса типа **C**, т. е. длинной строки текста, содержащий разделенные символом **«LF»** пункты вида **SERVICENAME|SERVICETYPE|SERVICECLASS**. Здесь **SERVICENAME** – имя для идентификации сервиса, **SERVICETYPE** – необязательный описатель типа (см. раздел 2.6), **SERVICECLASS** – класс сервиса (пустой для информационных сервисов, **CMD** для команд, **RPC** для удаленных процедур), а **«|»** - разделитель для полей. Например, DOZA/DEMO/SERVER/CLIENT_LIST|C| - информационный сервис с именем DOZA/DEMO/SERVER/CLIENT_LIST и типом C, а DIS_DNS/EXIT|L:1|CMD – командный сервис с именем DIS_DNS/EXIT и типом L:1.

Список клиентов **SERVERNAME/CLIENT_LIST** хранится в виде длинной строки текста, содержащий разделенные символом **«|»** пункты вида **PID@hostname**, где **PID** - идентификатор клиентского процесса, **hostname** – имя компьютера, на котором он запущен. Например, клиент 5028@ak-wxp-vm – это процесс 5028 на компьютере ak-wxp-vm.

Сервер имен **DNS** тоже является **DIM** сервером и имеет постоянное имя **DIS_DNS**. Он хранит список подключенных **DIM** серверов в виде сервиса типа **C** с постоянным именем **DIS_DNS/SERVER_LIST**. Имена серверов хранятся в **DNS** в виде длинной строки текста, содержащего список пунктов **SERVERNAME@hostname**, разделенных символом **«|»**. Здесь **SERVERNAME** – имя **DIM** сервера, **hostname** – имя компьютера, на котором этот сервер запущен. Этот список позволяет любому клиенту находить искомый **DIM** сервер по имени и определять имя компьютера, к которому надо подключаться. Например, DOZA/DEMO/SERVER@ak-wxp-vm - сервер DOZA/DEMO/SERVER на компьютере ak-wxp-vm.

При прекращении или возобновлении работы отдельного сервера или сервиса **DNS** посылает клиентам сообщение, в котором содержится

- **+SERVICENAME** – сервис возобновил работу
- **-SERVICENAME** – сервис прекратил работу
- **!SERVICENAME** – сервис недоступен

Поскольку первый символ сообщения маркирует статус сервера или сервиса, имена серверов и сервисов не должны содержать эти символы (**+**,**-**,**!**) в качестве первых символов.

Из описания того, как **DIM** хранит и передает списки серверов, клиентов и сервисов, становятся понятными перечисленные выше ограничения на их имена.

Следует отметить, что попытка запустить **DIM** сервер с дублирующимся именем или содержащим сервис с дублирующимся именем, который уже запущен и зарегистрирован в **DNS**, приведет к неудаче. Реакцией по умолчанию (если не указан обработчик таких событий) является завершение серверного процесса. Это служит защитой от попыток запуска одноименного **DIM** сервера и предохраняет **DIM** от ошибок, связанных с дублированием имен.

Наконец, заметим, что в одной физической сети может работать несколько **DIM** систем. Для того, чтобы они не конфликтовали друг с другом, достаточно использовать для каждой из них отдельный сервер имен **DNS**. Конфликтов имен не возникает потому, что каждый **DNS** будет обслуживать свое пространство имен. При этом запуск двух **DNS** на одном компьютере недопустим.

2.6 Соглашения по описанию типов

Описание типа сервиса не является обязательным, но (по возможности) настоятельно рекомендуется. Это описание полезно не столько для самого **DIM**, сколько для клиентов и отладочных утилит, позволяя им правильно интерпретировать данные сервиса.

Строка описания типа сервиса имеет вид следующего списка:

- **T:N;T:N;…;T[:N]**

где
T - описатель типа данных T=(C,D,F,I,L,S,X), см. таблицу 1
N - описатель длины данных, т. е. числа элементов типа T
: - разделитель для длины данных
; - разделитель записей

Последний пункт списка может не иметь описателя длины, что означает «открытый список». В этом случае число элементов данного типа определяется фактическим размером публикуемого сервиса.

Описатель типа может содержать такие значения:

Таблица 1. Описатели типа данных сервисов DIM.

Описатель типа	**Тип C/C++**	**Размер байт**	**Описание типа**
C	char	1	Символ, строка символов
D	double	8	Вещественное двойной точности
F	float	4	Вещественное одинарной точности
I	long int	4	Целое
L	Эквивалентно **I**	4	Целое
S	short int	2	Короткое целое
X	long long int	8	Длинное целое

Например, описатель типа **I:2;F:3;D:1;C** описывает структуру, содержащую { int[2], float[3], double[1], char[] }. Отсутствие описания длины в конце означает «открытый массив типа **C**», т. е. строку символов. Длина строки определяется фактическим размером данных, публикуемых сервером.

Если все элементы сервиса имеют один тип, используется простой символьный описатель типа, например **D** означает «открытый массив double[]». Длина массива определяется фактическим размером данных, публикуемых сервером.

Следует заметить, что в пакете **CRW-DAQ** [4,5] используются только типы (**C**,**D**,**I**), поэтому для согласования с пакетом использования других типов (**F**,**S**,**X**) следует избегать. Это не сильное ограничение, т. к. тип **F** является подмножеством **D**, а тип **S** - подмножеством **I**.

2.7 Сетевая адресация DIM

Обычно для клиент-серверной связи клиентам надо знать точные координаты каждого сервера (сетевой адрес, порт, протокол). Для больших систем управления это может оказаться серьезной проблемой, т. к. при большом числе серверов и клиентов на каждой клиентской машине должен храниться дубликат списка серверов. При миграции серверов на другие компьютеры эти списки надо каждый раз обновлять. Это затрудняет разработку и обслуживание систем.

В системе **DIM** этой проблемы нет. Для получения всей необходимой информации клиентам и серверам **DIM** надо знать только имя **DNS** сервера и имена интересующих сервисов. Как уже было сказано выше, сервер имен **DNS** (программа **DNS.EXE**) обеспечивает клиентам автоматический поиск нужного сервиса и обслуживающего его сервера. При миграции серверов и клиентов на другие машины ничего менять не требуется (если сервер имен **DNS** остается постоянным).

По умолчанию **DIM** берет имя **DNS** сервера из переменной окружения **DIM_DNS_NODE** (см. раздел 3.2). Поэтому перед вызовом многих **DIM** программ для них необходимо установить эту переменную окружения.

Если имя сервера **DNS** заранее неизвестно (например, вычисляется в процессе работы), то для его задания используются функции **DIM API** (см. раздел 4.2). Так или иначе, имя **DNS** должно быть задано первым, до вызова других клиент/серверных функций, т. к. при вызове этих функций имя **DNS** должно быть уже известно.

2.8 Потоковая модель DIM

Система **DIM** является многопоточной. Она создает служебные потоки, обслуживающие операции обмена данными и таймер. Вызов пользовательских процедур может производиться из контекста этих служебных потоков. Это значит, что пользовательские функции обратного вызова (callback) могут вызываться из другого (не основного) потока. Соответственно внутри пользовательских процедур, обслуживающих клиент-серверные операции, следует вызывать только потокобезопасные функции. При необходимости вызова функций, не являющихся потокобезопасными, необходимо использовать синхронизацию. Например, вместо перерисовки визуальных элементов нужно послать сообщение, которое выполнит требуемую прорисовку в основном потоке.

2.9 Временные параметры DIM

DIM имеет высокую скорость и малую задержку реакции - в пределах нескольких миллисекунд. Время реакции определяется в основном загруженностью процессора и частотой переключения потоков. Для уменьшения времени реакции есть возможность регулировать приоритет служебных потоков **DIM**, что позволяет снизить зависимость от загрузки процессора. При передаче по сети, естественно, играет также роль загруженность сети. Но это ограничение возникает при использовании любой другой сетевой технологии на основе **Ethernet**.

Практика показала, что **DIM** является хорошим выбором для систем мягкого реального времени с реакцией порядка 10 мс.

2.10 Производительность DIM

В 2005 г. проведены тесты производительности **DIM** на серверах:

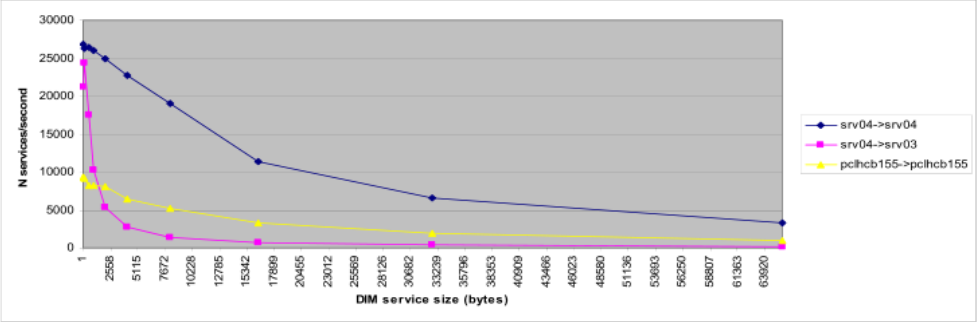
- **srv03** – Linux, Xeon Core2, 2.8 GHz, 1 GB RAM, Ethernet 100Mbit/s
- **srv04** – Linux, Xeon Core2, 2.8 GHz, 1 GB RAM, Ethernet 100Mbit/s
- **pclhcb155** – Windows XP, Pentium 4, 3 GHz, 1 GB RAM

Тестирование заключалось в непрерывном обновлении большого числа сервисов разного (фиксированного) размера. Измерялась частота (число обновлений в секунду) и поток (байт в секунду). При этом передача велась как по сети (клиент и сервер на разных машинах), так и в памяти (клиент и сервер на одной машине).

По результатам, приведенным на рисунке 6, по **DIM** передавалось (при достаточном размере сервиса) более 200 МБ/с в памяти и около 12.5 МБ/с по сети Ethernet-100, что является её естественным ограничением. При маленьком размере сервисов поток существенно снижается (что вполне естественно). Частота обновлений (для небольших сервисов) составляет от 10 до 25 тысяч обновлений в секунду, в зависимости от процессора. При увеличении размера сервисов частота снижается, т. к. ограничителем становится пропускная способность среды передачи.

Number of services received per second for different data sizes (in bytes):

	4	128	512	1024	2048	4096	8192	16384	32768	65536
srv04->srv04	26795	26362	26370	25980	24977	22703	19022	11398	6621	3226
srv04->srv03	21176	24399	17541	10214	5384	2794	1414	711	358	179
pclhcb155->pclhcb155	9254	9154	8253	8188	8048	6443	5182	3314	1933	1015



Throughput in Kbytes/second for different data sizes (in bytes):

	4	128	512	1024	2048	4096	8192	16384	32768	65536
srv04->srv04	104.7	3295	13185	25980	49954	90812	152176	182368	211872	206464
srv04->srv03	82.6	3049	8771	10214	10696	11176	11312	11376	11456	11456
pclhcb155->pclhcb155	36.1	1144	4127	8188	16098	25772	41456	53024	61856	64960

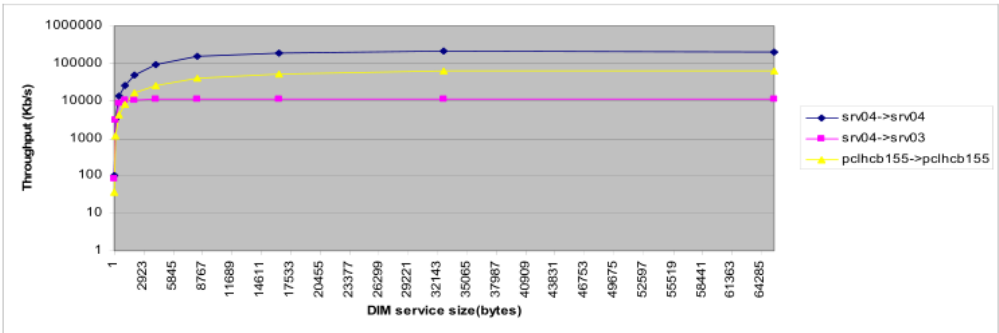


Рисунок 6 – результаты тестов производительности **DIM**.

Результаты измерений подтвердили высокую эффективность **DIM**.

2.11 Множественная конфигурация DIM

Как уже было сказано, система **DIM** может работать с несколькими серверами имен **DNS**, каждый из которых имеет свое пространство имен сервисов. Такие системы могут работать в одной физической сети, никак не мешая друг другу. Каждый **DNS** сервер обслуживает только те **DIM** серверы и клиенты, которые на него ссылаются. Каждый сервер имен **DNS** должен при этом располагаться на отдельном компьютере, повторный запуск **DNS** на том же компьютере невозможен.

Правда, надо заметить, что запуск двух и более серверов имен **DNS** все же возможен, но только в том случае, если эти **DNS** (как и все обслуживаемые им серверы и клиенты) сконфигурированы на разные порты (переменная окружения **DIM_DNS_PORT**, см. раздел 3.2).

Кроме случая изолированных сетей работа **DIM** возможна также в закрытых сетях и на машинах с несколькими сетевыми интерфейсами.

Сети **DIM**, имеющие разные **DNS**, могут не только работать автономно, но и передавать данные друг другу с помощью «мостов» **DimBridge**, см. раздел 3.5 .

Объединять сети **DIM** с помощью мостов можно по-разному. На рисунке 7 показано, как можно объединить сети **DIM** парными мостами. Такой способ пригоден для небольших систем, где число парных связей невелико.

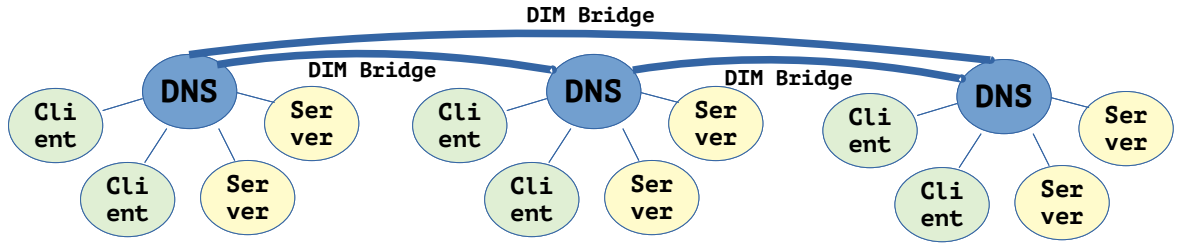


Рисунок 7 – Объединение сетей **DIM** с помощью парных мостов.

Для крупных сетей более подходящим будет объединение **DIM** сетей с помощью центрального **DNS** сервера. В этом случае число мостов равно числу сетей **DIM**, но при этом требуется выделенный сервер для **DNS**.

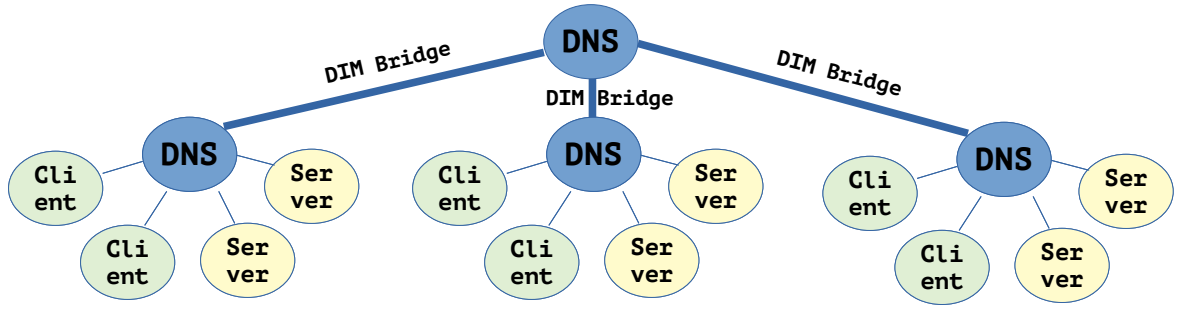


Рисунок 8 – Объединение сетей **DIM** с помощью центрального **DNS**.

В обоих случаях отдельные **DIM** подсистемы могут продолжать работу автономно, даже если связь между сетями прерывается. Использование такой схемы уместно, если, например, эти подсистемы требуют отказоустойчивой работы и не могут зависеть от центрального сервера.

2.12 Заметки и Примечания

Следует сделать несколько пояснительных заметок и примечаний, которые позволят избежать ненужной путаницы в терминах:

- Не следует путать сервер имен **DIM DNS** (dynamic name server) с доменным сервером имен **DNS** (domain name server). Они решают совершенно разные (хотя и похожие) задачи. Сервер имен **DIM DNS** сопоставляет имена **DIM** сервисов и **DIM** серверов. Доменный сервер сопоставляет имена компьютеров и **IP** адреса. Сервер имен **DIM DNS** при работе опирается (в том числе) на доменный сервер имен **DNS**, чтобы разрешать **IP** адреса (находить адреса компьютеров по их именам). Одинаковая аббревиатура **DNS** может вводить в заблуждение, поэтому всегда надо обращать внимание на контекст обсуждения, чтобы понять, о каком **DNS** идет речь.
- Сервер **DIM** не следует путать с сервером в смысле «компьютер». **DIM** сервер – это **процесс**, предоставляющий сервисы по протоколу **DIM**. На одном компьютере может работать несколько **DIM** серверов. Имя **DIM** сервера – это не имя компьютера, а имя процесса, под которым он зарегистрировался в **DIM DNS** при вызове функции **dis_start_serving** (см. раздел 4.2). Сервер имен **DIM DNS** позволяет транслировать имя **DIM** сервера в имя компьютера, на котором он запущен. Это делается автоматически при подключении клиентов и не должно заботить пользователя.

2.13 Недостатки DIM

Картина будет неполной, если не отметить (возможные) недостатки **DIM**. Тут надо сказать, что рассматривать какие-либо свойства системы как недостаток можно только в связи с условиями её эксплуатации. В других условиях эксплуатации эти недостатки могут не проявляться.

Одним из возможных недостатков **DIM** является защита информации. В системе **DIM** нет системы авторизации доступа и шифрования данных. Все сервисы доступны всем клиентам. **DIM** может лишь ограничить доступ на уровне списка разрешенных компьютеров, но не более того. Предполагается, что **DIM** работает в закрытой сети или в сети, которая уже защищена той или иной **СЗИ**.

Этот недостаток не является абсолютным, т. к. ничто не мешает защищать сервисы шифрованием на уровне пользователя. При этом зашифрованные сервисы можно передавать по открытым каналам без риска несанкционированного доступа. Именно такой подход был реализован при интеграции системы **DIM** в пакет **CRW-DAQ**, где шифрование содержимого **DIM** сервисов используется для организации системы авторизации доступа к управлению.

Другим возможным недостатком является наличие центрального **DNS**. Если не обеспечить его бесперебойную работу, то клиенты и серверы не смогут связаться (даже при наличии парной связи). Для случая децентрализованной сети, где нет выделенного сервера **DNS**, больше подойдет работа с несколькими локальными **DNS**, связанными сетевыми мостами, см. раздел 3.5 . Однако следует заметить, что это больше забота сетевых администраторов, чем пользователей. В любом случае клиенты и серверу пишутся одинаково, а вопросы организации сети решаются путем конфигурации сетевых служб.

2.14 Некоторые рекомендации по проектированию

Технология **DIM** разработана для применения в больших проектах, где участвует много независимых групп разработчиков. При этом для совместной работы им надо выработать и вместе согласовать договоренности об обмене данными. Технология **DIM** определяет общую «рамку соглашений», которую всегда приходится дополнять конкретной информацией, индивидуальной для каждой системы.

Здесь дается несколько простых рекомендаций по проектированию **DIM** серверов и клиентов. Это важный вопрос, т. к. проектирование включает в себя выработку спецификации (договоренности) о составе сервисов, которую разработчик **DIM** сервера предоставляет всем пользователям (клиентам) еще на этапе согласования совместных работ. Здесь важно не ошибиться в определении состава и структуры публикуемых сервисов **DIM**, т. к. изменение спецификации после согласования нежелательно из-за проблем, которые эти изменения создают в больших проектах, где работает много участников, которые используют совместно выработанные спецификации для обмена данными.

Спецификация **DIM** сервера, создаваемая разработчиком **DIM** сервера и согласуемая с его пользователями (клиентами) должна включать в себя как минимум:

1. Общую информацию о сервере (имя **DIM** сервера; используемые порты, если они отличаются от стандартных; краткое описание основных функций сервера).
2. Соглашения о наименовании сервисов (общие правила).
3. Таблицу публикуемых сервисов, включающую **имя**, **тип** и **описание** каждого сервиса.

Пример спецификации DIM сервера приведен в Приложении 7.

С точки зрения производительности (см. раздел 2.10) важно правильно сгруппировать данные в **DIM** сервисы. При небольшом размере данных сервисов (примерно от 1 байта до 1 килобайта) максимальная частота обновлений сервисов практически не зависит от размера сервиса. Это значит, что данные выгодно группировать, помещая в один сервис несколько элементов данных, т. к. с точки зрения производительности нет разницы, например, в передаче 1 или 100 целых или вещественных чисел. Поэтому чем больше данных передается за одно обновление, тем выше общая производительность, и для передачи нужных данных потребуется меньше актов обновления.

Однако при группировании данных в сервисы следует учитывать также **время** обновления. Если данные, находящиеся в одном **DIM** сервисе, обновляются в разное время, то придется обновлять сервис многократно, т. к. обновление данных сервисов обычно делается по факту изменения сервиса (т. е. любого из его элементов).

По этой причине следует проектировать **DIM** сервисы так, чтобы в один сервис по возможности помещались все данные, изменяющиеся **одновременно**. Например, это могут быть данные каналов одного регистратора (если они поступают одним пакетом). В этом случае группировка таких одновременно обновляемых данных приведет к оптимизации трафика и числа необходимых обновлений сервисов.

3 Практические вопросы использования DIM

В данном разделе приводятся полезные практические сведения по использованию и настройке **DIM** в **ОС Windows**.

3.1 Необходимые файлы времени исполнения

В таблице 2 приведен минимальный список файлов из каталога **dim\bin**, необходимых для запуска любого **DIM** клиента или сервера (при использовании динамической компоновки). При этом необходимо обеспечить, чтобы эти файлы были доступны запускаемой программе (т. е. или скопированы в каталог программы, или доступны через переменную окружения **PATH**).

Таблица 2. Минимальный список файлов для клиентов и серверов **DIM**.

Файлы	**Описание**
DIM.DLL	Основная библиотека **DIM**.
msvcm80.dll **msvcp80.dll** **msvcr80.dll** **Microsoft.VC80.CRT.manifest**	Файлы библиотеки времени исполнения компилятора **MS Visual C**, под которым компилировался **DIM.DLL**. Этот набор зависит от версии компилятора и может меняться в зависимости от версии **DIM**. Данная библиотека актуальна для используемой версии **DIM 20.29**.

В таблице 3 приведен список основных инструментальных программ, которые (в принципе) нужны для работы с системой **DIM**. Эти программы не обязательно должны присутствовать на клиентских машинах. Например, сервер имен **DNS** запускается только на одной машине в сети, а межсетевой мост и отладочные утилиты используются только в специальных случаях (для настройки и отладки).

Таблица 3. Список основных инструментальных программ **DIM**.

Файлы	**Описание**
DNS.EXE	**Сервер имен**. Должен быть **постоянно** запущен на машине, которая избрана сервером имен. На остальных машинах не нужен.
DIMBRIDGE.EXE	**Межсетевой мост**. Служит для связи нескольких сетей **DIM** с разными **DNS** друг с другом. Нужен только при использовании нескольких серверов имен **DNS**.
DID.EXE	Отладочная утилита. Позволяет просматривать все доступные сервера и их сервисы.
DIMTREE.EXE	Отладочная утилита. Дает удобный древовидный интерфейс для просмотра и анализа всех доступных **DIM** серверов и публикуемых ими сервисов и данных.

Все остальные файлы - тестовые и демонстрационные утилиты, документация, программные интерфейсы и исходные коды - нужны только в помощь разработчикам и отладчикам.

3.2 Переменные окружения

В таблице 4 указаны переменные окружения, используемые для конфигурации клиентов и серверов **DIM**. Как правило, эти переменные используются для задания параметров в том случае, если они не были заданы явным вызовом функцией **DIM API**, см. раздел 4.2 .

Таблица 4. Переменные окружения для конфигурации **DIM**.

Переменная	**Описание**
DIM_DNS_NODE **или вызовы** **dim_set_dns_node()** **dic_set_dns_node()** **dis_set_dns_node()**	Перед запуском Клиента или Сервера им необходимо указать имя компьютера, на котором работает сервер имен **DNS**. Указание **DNS** строго обязательно (значение по умолчанию не определено). Переменная окружения не используется, если **DNS** задан явно (функцией **DIM API**, см. раздел 4.2).
DIM_DNS_PORT **или вызовы** **dic_set_dns_port()** **dic_set_dns_port()** **dis_set_dns_port()**	Перед запуском Клиента или Сервера им необходимо указать порт **DNS** (по умолчанию **2505**). Использование разных портов дает возможность запускать несколько **DNS** на одной машине.
DIM_HOST_NODE	Перед запуском Сервера можно указать, какое **IP** имя или адрес нужно использовать клиентам для подключения к серверу. Допускается **IP** имя (типа `pclhcb155.cern.ch`) или **IP** адрес (типа `137.138.214.208`). По умолчанию используется имя, которое возвращает команда **hostname**. Другое имя задается, например, если Сервер доступен Клиентам через маршрутизатор.
DIM_DNS_ACCEPTED_DOMAINS	Перед запуском сервера имен **DNS** задает разделенный запятыми список допустимых доменов, с которыми будет работать **DNS**. По умолчанию разрешены все домены. Используется только в **DNS**.
DIM_DNS_ACCEPTED_NODES	Перед запуском сервера имен **DNS** задает разделенный запятыми список допустимых узлов (машин), с которыми будет работать **DNS**. По умолчанию разрешены все узлы. Используется только в **DNS**.
DIM_KEEPALIVE_TMOUT	Задает время ожидания (в секундах, по умолчанию **15**) для попытки установить соединение с сервером, прежде чем будет зафиксирована ошибка соединения.
DIM_WRITE_TMOUT	Задает время ожидания (в секундах, по умолчанию **5**) для попытки записи (передачи данных по каналу связи), прежде чем будет зафиксирована ошибка записи.

3.3 Используемые DIM порты TCP/IP

Для связи **DIM** использует (**1 + 3 * N**) портов **TCP**, где **N** – число запущенных на машине **DIM** серверов. Порты перечислены в таблице.

Таблица 5. Порты **TCP**, используемые **DIM** для связи.

Порт и способ задания	**Описание**
2505 **DIM_DNS_PORT** **dim_set_dns_node()** **dic_set_dns_port()** **dis_set_dns_port()**	По умолчанию, или через переменную окружения, или прямым заданием для клиента/сервера, или прямым заданием для клиента и прямым заданием для сервера - задает порт **TCP**, используемый для связи с сервером имен **DNS**.
5100, **5101**, **5102**, … до **6000** (выбор автоматический)	Порты для приема/передачи данных **DIM**. Используется по три порта на **DIM** сервер. Свободные порты выбираются автоматически.

Для работы **DIM** указанные порты обязательно должны быть разрешены в Сетевом Экране (обычно это **Windows Firewall**). Командный файл, с помощью которого можно быстро разрешить, запретить и посмотреть правила фильтрации для **DIM** портов, приведен в приложении 6.

При работе в сетях с сетевым шлюзом (gateway), маршрутизатором (router) или аппаратным сетевым экраном (firewall) указанные порты должны быть разрешены и соответствующим образом настроены.

3.4 Сетевые службы DIM

Для работы **DIM** на одном из компьютеров в сети должна непрерывно работать программа «сервер имен» **DNS**, иначе серверы и клиенты не смогут найти друг друга. Следовательно, сервер **DNS** должен быть постоянно включен, а программа **DNS** должна автоматически запускаться и восстанавливаться при «падении». Разработчики должны обеспечить это условие.

При наличии нескольких **DNS** и «мостов» между ними (см. раздел 3.5) необходимо также обеспечить бесперебойную работу этих «мостов».

Все остальные серверы и клиенты могут работать в свободном режиме. Серверы и клиенты могут включаться, выключаться, мигрировать с машину на машину – связь будет восстанавливаться автоматически в кратчайшее время, как только это станет возможным.

В пакете **CRW-DAQ** интегрированный **DIM** сервер **&DimSrv** всегда запускает локальный **DNS.EXE** (в скрытом консольном окне). Он может использоваться или игнорироваться (накладные расходы при этом ничтожны), однако он всегда доступен. Сервер **&DimSrv** также имеет службу поддержки «мостов» **DimBridge**, если они используются. Поэтому каждая машина с работающим пакетом **CRW-DAQ** и сервером **&DimSrv** (потенциально) может служить **DNS** сервером, как и сервером «межсетевых мостов».

3.5 Сетевой мост DIM Bridge

При наличии нескольких сетей **DIM** с разными **DNS** для связи между ними используется «сетевой мост», передающий сервисы от одного **DNS** другому. Возможен случай, когда **DNS** работает в двух физических сетях (на машине с двумя сетевыми картами). В этом случае данные могут передаваться по «мосту» между физически разными сетями.

Программа «сетевой мост» **DimBridge** может передавать **DIM** сервисы и команды от одного **DNS** другому **DNS**, связывая две сети «мостом». Эти **DNS** могут находиться даже в разных физических сетях (если один из **DNS** работает сразу в двух сетях). Имейте в виду, что при большом потоке данных **DimBridge** может стать «узким местом», т. к. весь поток данных между двумя сетями будет проходить через него.

Использование:

DimBridge [from_node] to_node services

где:
from_node – первый **DNS** (источник), по умолчанию **DIM_DNS_NODE**
to_node – полное сетевое имя второго **DNS** (приемника)
services – список имен сервисов (маски имен поддерживаются)

Например:

DimBridge source.this.net target.other.net DEMO/SERVICES/*

В силу реализации **DimBridge** понимает только полные сетевые имена (типа host.domain) и выдает ошибку на простые имена (типа localhost). Если в сети нет домена, в конце имени машины-источника надо ставить точку (localhost.), имитируя полное сетевое имя.

Мост **DimBridge** может выполнять много разных задач. Например, если надо передавать данные в/из закрытой сети, это можно сделать через шлюзовую машину с двумя сетевыми картами, на которой работает мост, передающий нужные сервисы в другую **DIM** сеть.
Другим примером является использование моста **DimBridge** в системах, где нет постоянно работающего центрального сервера **DNS**. В этом случае можно в каждом сегменте сети (или даже на каждом компьютере) запускать свой локальный **DNS** и связывать их мостами. В таком случае зависимость от центрального **DNS** исчезает, а каждая локальная группа может работать бесперебойно со своим сервером **DNS** в автономном режиме или со связью с другими группами.

Рекомендованная практика именования **DIM** серверов (с префиксом **DIM_TASK**) помогает отделить информационные сервисы **DIM** от служебных, что позволяет организовать экспорт всех данных сервера через мост **DimBridge** по маске имен информационных сервисов.

В пакете **CRW-DAQ** интегрированный **DIM** сервер **&DimSrv** имеет службу поддержки «сетевых мостов» (команда **@DimBridge**), обеспечивая их автоматический запуск при старте и перезапуск при сбое.

4 Программный интерфейс DIM API

Программный интерфейс **DIM API** доступен для множества популярных языков программирования (**C**, **C++**, **Fortran**, **Pascal**, **Java**, **LabView**) и включает в себя, кроме исходного кода, исполняемые библиотеки (основная из них **DIM.DLL**), заголовочные файлы, дающие доступ к библиотекам, инструментальные программы (сервер имен **DNS**, отладочные утилиты), примеры, документацию, включающую соглашения о наименовании серверов, сервисов, и соглашения об описании типов.

4.1 Обзор состава библиотек DIM

Система **DIM** доступна в исходных кодах (на **C**, **C++**, **Java**), которые можно взять на сайте http://dim.web.cern.ch. Он поставляется в архиве **ZIP** с именем, зависящим от версии. Например, **dim_v20r2.zip**. Объем архива варьируется от 3 до 10 МБ, в зависимости от версии.

В пакете **CRW-DAQ** [4,5] библиотеки **DIM** расположены в каталоге **DimSite** и содержат, кроме оригинальных файлов **DIM** в каталоге **dim**, дополнительные файлы – интерфейсы для языков программирования, демонстрационные примеры, документацию. Наиболее важные каталоги и файлы перечислены в таблице 6.

Таблица 6. Некоторые файлы из состава дистрибутива **DIM**.

Каталог/файл	**Комментарий**
dim\	Каталог дистрибутива DIM, с сайта [1]
dim\bin\	Каталог исполняемых файлов DIM (exe,dll)
dim\bin\dim.dll	Основная библиотека DIM.DLL, см. раздел 3.1
dim\bin\DID.exe	Утилита DID для отладки, см. раздел 2.4
dim\bin\DIMTree.exe	Утилита DIMTREE для отладки, см. 2.4
dim\bin\dns.exe	Сервер имен DNS, см. раздел 2.2
dim\bin\DimBridge.exe	Сетевой мост DIM Bridge, см. раздел 3.5
dim\dim\	Каталог заголовочных файлов для C/C++
dim\dim\dic.h	Заголовочный файл клиента DIM
dim\dim\dim.h	Заголовочный файл общих объектов DIM
dim\dim\dis.h	Заголовочный файл сервера DIM
dim\jdim\	Каталог библиотек Java
dim\src\	Каталог исходных кодов DIM
dim\Visual	Каталог проектов Visual Studio
dim_dcc	Каталог библиотек для Pascal (Delphi)
dim_dcc_DIM.PAS	Заголовочный файл для Pascal (Deplhi)
dim_demo	Каталог с демонстрационным примером
dim_demo\client.dpr	Демонстрационный клиент (исходный код)
dim_demo\client.exe	Демонстрационный клиент (исполняемый файл)
dim_demo\demo.bat	Демонстрационный пример (команда запуска)
dim_demo\make.bat	Демонстрационный пример (команда компиляции)
dim_demo\server.dpr	Демонстрационный сервер (исходный код)
dim_demo\server.exe	Демонстрационный сервер (исполняемый файл)
dim_docs	Документация DIM на русском
dim_manual.pdf	Документация DIM на английском
dim_overview.pdf	Статья с описанием DIM на английском

Полный список файлов архива **DIM** приведен в Приложении 1.

20.12.2021

4.2 Обзор программного интерфейса DIM API

Обзор программного интерфейса **DIM** будет проводиться на примере языка **Pascal** (**Delphi**), исходный код интерфейсного модуля **_DIM.PAS** для которого приведен в Приложении 2. Интерфейсы для других языков устроены аналогично. Фактически это заголовочный файл для **DIM.DLL**.

Программный интерфейс **DIM API** устроен довольно просто и логично. Он активно использует пользовательские процедуры, обратного вызова (**callback**), которые вызываются при наступлении определенных событий, например, приема данных от сервера. Собственно, эти пользовательские функции обычно и выполняют содержательную работу, предоставляя исходные данные для сервера и выполняя прием данных на стороне клиента. **DIM** обеспечивает все остальное – поиск сервера, подключение к нему, поддержание сеанса связи, прием и передачу данных по сети. Все эти сложности скрыты от пользователя.

Следует заметить, что для в идентификаторах типов, объектов и функций используются такие сокращения:

- dim - общие для клиента и сервера функции **DIM**
- dis - функции сервера (distributed information server)
- dic - функции клиента (distributed information client)
- dns - функции сервера имен **DNS** (dynamic name server)
- dtq - функции таймера (DIM timer queue)

Таблица 7. Некоторые функции программного интерфейса **DIM API**.

Идентификатор	**Комментарий**
TDis_User_Routine	Тип callback процедуры информационного сервера
TDis_Cmnd_Routine	Тип callback процедуры командного сервера
TDis_Exit_Handler	Тип callback процедуры завершения сервера
TDic_User_Routine	Тип callback процедуры информационного клиента
TDic_Cmnd_Routine	Тип callback процедуры командного клиента
TDtq_User_Routine	Тип callback процедуры обработчика таймера
dim_init	Инициализация DIM
dim_set_dns_node	Установка имени DNS сервера
dim_get_dns_node	Чтение имени DNS сервера
dis_disable_padding	Режим выравнивания адресов для сервера
dis_add_service	Добавление информационного сервиса в сервер
dis_remove_service	Удаление сервиса из сервера
dis_update_service	Обновление информационного сервиса сервером
dis_add_cmnd	Добавление командного сервиса в сервер
dis_start_serving	Начало обслуживания для сервера
dis_stop_serving	Остановка обслуживания для сервера
dis_get_client	Чтение имени подключенного к серверу клиента
dic_disable_padding	Режим выравнивания адресов для клиента
dic_info_service	Добавление информационного сервиса к клиенту
dic_cmnd_service	Добавление командного сервиса к клиенту
dic_release_service	Удаление сервиса из клиента
dic_get_format	Чтение строки описания формата сервиса
dic_close_dns	Закрыть соединение с DNS

Полный набор функций (около 130) программного интерфейса **DIM API** приведен в Приложении 2.

4.3 Структура типичного DIM сервера

Для создания типичного **DIM** сервера следует выполнить примерно следующие операции:

- Инициализировать **DIM** вызовом dim_init, dis_disable_padding, …
- Установить обработчики событий ошибок, выхода вызовом функций dis_add_error_handler, dis_add_exit_handler и т.д.
- Установить имя **DNS** сервера вызовом dim_set_dns_node, чтобы указать серверу, к какому серверу имен **DNS** надо подключаться.
- Добавить к серверу информационные и командные сервисы вызовом dis_add_service, dis_add_cmnd. При этом надо указать имя и тип сервиса, а также написать пользовательские процедуры (callback), которые будет вызываться при обновлении данных и получении команд от клиентов.
- Запустить обслуживание вызовом dis_start_serving. При этом указывается имя сервера, под которым он будет известен в **DIM**.
- В цикле опроса, по мере поступления новых данных, следует вызывать процедуры dis_update_service для обновления сервисов. Нужные пользовательские callback процедуры для обновления данных будут вызваны автоматически.
- При завершении работы освободить все сервисы вызовом dis_remove_service, а затем завершить обслуживание вызовом dis_stop_serving.

К преимуществам **DIM** можно отнести то, что серверу совсем не надо заботиться о клиентских подключениях – всю эту работу выполняет **DIM**. Сервер работает одинаково, независимо от числа подключенных клиентов – он просто выполняет обновление сервисов, как только это становится нужным. Обслуживание клиентских подключений и рассылку всем клиентам обновленных данных осуществляет сам **DIM**, без усилий со стороны прикладного программиста.

Другим преимуществом является возможность отладки сервера с помощью готовых утилит **DID**, **DIMTREE**, **DIMSTAT**. Программисту достаточно проверить работу сервера на этих утилитах, чтобы быть уверенным, что любой клиент тоже сможет работать с этим сервером.

При написании пользовательских callback процедур обработки событий следует учитывать, что вызов этих процедур может осуществляться в контексте другого (служебного) потока, поэтому необходимо позаботиться о потоковой безопасности этих процедур. Например, для рисования нужно посылать основному потоку сообщение на прорисовку, т. к. в служебном потоке вызывать функции рисования будет не безопасно.

Надо также отметить, что при написании пользовательских callback процедур обработки используется целочисленный параметр **tag**, который служит для идентификации сервиса. Этот параметр указывается при регистрации сервиса, сохраняется в системе **DIM** и передается без изменений во все callback процедуры при их вызове. Часто в этом параметре хранится и передается указатель на объект, представляющий сервис, при этом используется преобразование типа.

Пример демонстрационного DIM сервера приведен в Приложении 3. Для запуска демонстрационного примера используется командный файл, приведенный в Приложении 5.

4.4 Структура типичного DIM клиента

Для создания типичного **DIM** клиента следует выполнить примерно следующие операции:

- Инициализировать **DIM** вызовом dim_init, dic_disable_padding, …
- Установить обработчики событий ошибок, выхода вызовом функций dic_add_error_handler, dic_add_exit_handler и т.д.
- Установить имя **DNS** сервера вызовом dim_set_dns_node, чтобы указать клиенту, к какому серверу имен **DNS** надо подключаться.
- Добавить к клиенту информационные сервисы вызовом функции dic_info_service. При этом надо указать имя сервиса, а также написать пользовательские процедуры (callback), которые будут вызываться при получении данных от сервера. Обслуживание начинается немедленно после добавления клиентского сервиса.
- В цикле опроса, по мере необходимости, следует вызывать процедуры dic_cmnd_service для посылки команд серверу. Как правило, команды являются реакцией на данные, получаемые от сервера и служат для изменения его состояния.
- При завершении работы надо освободить все сервисы вызовом dic_release_service, а затем завершить обслуживание вызовом dic_close_dns.

К преимуществам **DIM** можно отнести то, что клиенту совсем не надо заботиться о подключении к серверу – всю эту работу выполняет **DIM**. Клиент работает одинаково, независимо от числа и расположения используемых серверов – он просто выполняет подписку на сервисы, которые его интересуют. Обслуживание сетевых подключений и получение клиентом обновленных данных от сервера осуществляет сам **DIM**, без усилий со стороны прикладного программиста.

Другим преимуществом является возможность отладки клиентов с помощью готовых утилит **DID**, **DIMTREE**, которые могут работать и как тестовые серверы. Программисту достаточно проверить работу клиента на этих утилитах, чтобы быть уверенным, что любой другой сервер тоже сможет работать с этим клиентом.

При написании пользовательских callback процедур обработки событий следует учитывать, что вызов этих процедур может осуществляться в контексте другого (служебного) потока, поэтому необходимо позаботиться о потоковой безопасности этих процедур. Например, для рисования нужно посылать основному потоку сообщение на прорисовку, т. к. в служебном потоке вызывать функции рисования будет не безопасно.

Надо также отметить, что при написании пользовательских callback процедур обработки используется целочисленный параметр **tag**, который служит для идентификации сервиса. Этот параметр указывается при регистрации сервиса, сохраняется в системе **DIM** и передается без изменений во все callback процедуры при их вызове. Часто в этом параметре хранится и передается указатель на объект, представляющий сервис, при этом используется преобразование типа.

Пример демонстрационного **DIM** клиента приведен в Приложении 4. Для запуска демонстрационного примера используется командный файл, приведенный в Приложении 5.

4.5 Библиотеки классов для DIM

В разделах (4.1 … 4.4) был кратко рассмотрен интерфейс **DIM API** для классических (процедурных) языков программирования. Этот интерфейс наиболее переносимый между различными языками программирования и операционными системами. Однако не все программируют только на процедурных языках, многим больше нравятся классы и **ООП**.

Программирование клиентов и серверов для **DIM** может стать проще, если использовать библиотеки классов. В состав оригинального дистрибутива **DIM** входят библиотеки классов на языках **C++** и **Java**. По ним имеется подробная документация (на сайте [1]), поэтому они здесь не рассматриваются.

Для программирования на **Object Pascal** была (дополнительно) создана своя библиотека классов, включающая файлы **_DIM.PAS** (общая интерфейсная библиотека для **DIM.DLL**, см. Приложение 2), **_DIMC.PAS** (клиентская библиотека), **_DIMS.PAS** (серверная библиотека). Кроме того, в сборку входит каталог **dim_democ**, где содержится пример программирования клиента и сервера с использованием этих библиотек. Использование библиотек классов является в ряде случаев удобным, но не обязательным, т. к. все задачи можно решить с помощью функций **DIM API** из файла **_DIM.PAS**. Сама библиотека классов является «надстройкой» над этим интерфейсом. Из-за большого объема эта библиотек здесь не приводятся, однако надо принять к сведению, что она есть и может пригодиться при создании приложений. Заметим, что программа **DIMTREE.EXE** написана с использованием этой библиотеки классов, поэтому её исходный код также может служить примером использования. Все технические подробности можно найти непосредственно в файлах дистрибутива.

4.6 Пользовательские процедуры (callback)

Система **DIM**, с точки зрения программиста, базируется на пользовательских процедурах (**callback**), вызываемых при наступлении определенных событий: прием и передача данных, обработка ошибок, подключение и отключение клиентов и т.д. При написании этих процедур следует придерживаться нескольких простых правил:

- Помнить о потоковой безопасности, т. к. вызов процедур может происходить в контексте служебного потока. Для выполнения требуемой работы в основном потоке используется синхронизация (сообщения, очереди, общая память и другие методы).
- Исключения, возникающие в процедурах, следует обрабатывать внутри этих процедур, т. к. необработанные исключения могут нарушить работы библиотеки **DIM**, написанной без использования структурной обработки исключений (**SEH**).
- Процедурам, как правило, передается параметр **tag** целого типа, идентифицирующий обрабатываемый сервис. Хотя **tag** передается как переменная (это сделано для совместимости с другими языками), его значение нельзя менять. В параметре **tag** можно передавать указатель на сервис, при этом используется явное преобразование типов. Значение **tag** задается при вызове процедуры регистрации сервиса (например, **dis_add_service**) и после этого не меняется.

Содержательные примеры процедур обработки событий **DIM** можно найти в демонстрационных программах в Приложениях 3, 4.

Заключение

Сетевая технология **DIM** хорошо приспособлена для создания распределенных измерительных систем и систем управления (мягкого) реального времени в смешанной среде, состоящей из разных компьютеров, работающих на разных аппаратных и программных платформах.

Хорошая переносимость и поддержка большого числа операционных систем и языков программирования позволяет легко увязывать интересы участников и организовывать работу больших экспериментов или установок, в которых разные группы разработчиков могут программировать на разных языках и использовать разные аппаратные и программные платформы.

Хорошая масштабируемость, низкие аппаратные требования, высокая эффективность работы позволяют одинаково эффективно использовать **DIM** как в малых (1-2 компьютера), так и в больших (сотни компьютеров) системах.

Открытый исходный код и свободная лицензия (**GPL**) позволяет использовать **DIM** в самых разных задачах, не накладывая ограничений на её распространение.

Сетевая прозрачность, отказоустойчивое восстановление связи после сбоев или миграции серверов на другие компьютеры позволяет использовать **DIM** в системах, требующих высокой надежности и возможности «горячего» резервирования.

Наличие готовых универсальных утилит для наблюдения и отладки позволяет сделать разработку распределенных систем управления на основе **DIM** удобной и высокоэффективной.

Перечисленные достоинства позволяют автору рекомендовать **DIM** как одну из лучших коммуникационных технологий связи для решения задач автоматизации измерительных систем и физических установок.

Список использованных источников

1. http://dim.web.cern.ch/dim/

2. C.Gaspar, M.Dönszelmann, Ph.Charpentier. DIM, a portable, light weight package for information publishing, data transfer and inter-process communication. Computer Physics Communications, Volume 140, Issues 1-2, 15 October 2001, Pages 102-109.

3. http://cern.ch

4. А.В.Курякин, Ю.И.Виноградов. Программа для автоматизации физических измерений и экспериментальных установок (CRW-DAQ). // Свидетельство РФ об официальной регистрации программы для ЭВМ № 2006612848 от 10.08.2006 г. Сайт программы: www.crw-daq.ru

5. А.В. Курякин, Ю.И. Виноградов. Программное обеспечение автоматизированных измерительных систем в области тритиевых технологий. // ВАНТ, серия «Термоядерный синтез», 2008 г., выпуск 2, стр.80-90.

6. OLE for Process Communication. http://www.opcfoundation.com

7. The ALICE Collaboration (K. Aamodt et al). The ALICE experiment at the CERN LHC. 2008 JINST 3 S08002.

8. A.Kurepin, A.Augustinus et al. ALICE DCS preparation for RUN 3. Proceedings of the VIII International Conference "Distributed Computing and Grid-technologies in Science andEducation" (GRID 2018), Dubna, Moscow region, Russia, September 10 - 14, 2018, p.65-69.

9. ALICE Collaboration (S. Acharya et al). The ALICE Transition Radiation Detector: Construction, operation, and performance. Nuclear Inst. and Methods in Physics Research, A 881 (2018) 88-127.

10. А.Н.Курепин. Автоматизированная система управления и контроля стартового детектора времяпролетной системы эксперимента ALICE на Большом Адронном Коллайдере. Диссертация на соискание ученой степени кандидата физико-математических наук. Москва 2014.

11. А.В.Курякин, Ю.И.Виноградов, Н.В.Завьялов и др. Опыт длительной эксплуатации автоматизированной системы охлаждения электромагнитного калориметра PHOS в эксперименте ALICE. Ядерная физика и инжиниринг, 2012, том 3, № 6, с. 540-551.

12. https://smi.web.cern.ch/ - SMI++ home web site.

13. C. Gaspar and B. Franek, Tools for the automation of large distributed control systems, IEEE Trans. Nucl. Sci. 53 (2006) 974.

14. B. Franek and C. Gaspar, SMI++ – Object Oriented Framework for Designing and Implementing Distributed Control Systems, IEEE Trans. Nucl. Sci. 51 (2004) 513.

15. A. Barriuso Poy et al. The detector control system of the ATLAS experiment. 2008 JINST 3 P05006.

16. K. Lantzsch et al. The ATLAS Detector Control System. International Conference on Computing in High Energy and Nuclear Physics 2012 (CHEP2012). Journal of Physics: Conference Series 396 (2012) 012028.

17. The LHCb Collaboration (D. Esperante, P. Rodríguez et al). LHCb silicon tracker DAQ and DCS online systems. 2009 16th IEEE-NPSS Real Time Conference - RT'09.

18. E. Cortina Gil et al. The beam and detector of the NA62 experiment at CERN. 2017 JINST 12 P05025.

19. B. Copy, E. Mandilara, I. Prieto Barreiro, F. Varela Rodriguez. Monitoring of CERN's data interchange protocol (DIP) system. 16th Int. Conf. on Accelerator and Large Experimental Control Systems - ICALEPCS2017, Barcelona, Spain. doi:10.18429/JACoW-ICALEPCS2017-THPHA162, p.1797-1800.

20. M. Gonzalez-Berges. The Joint COntrols Project Framework. Computing in High Energy and Nuclear Physics, March 24-28 2003, La Jolla, California.

21. William Badgett, Laura Borrello et al. Web Based Monitoring in the CMS Experiment at CERN.

22. Bert Petersen. The cosmic ray induced muon spectrum measured with the L3 detector. Radboud University Nijmegen 2002.

23. K. Weber et al. Data Acquisition System of the CLOUD Experiment at CERN. IEEE transactions on instrumentation and measurement,vol.70,2021.

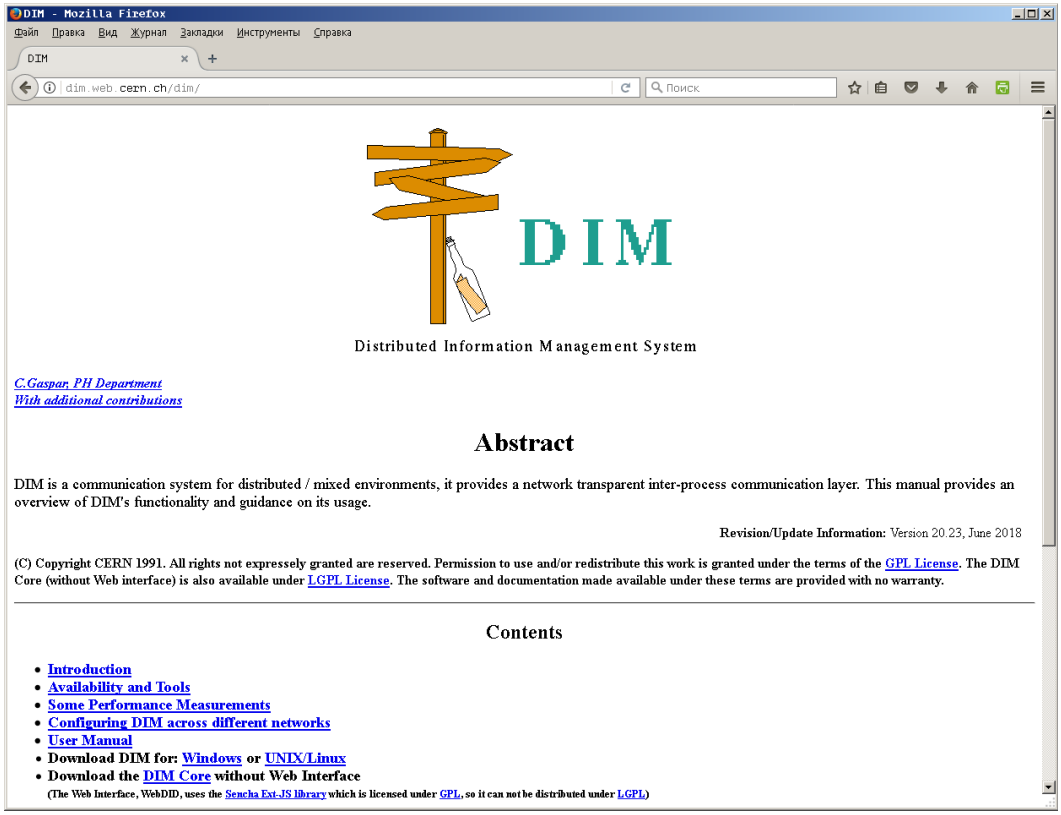


Рисунок 9 – Домашний сайт **DIM**.

Приложение 1. Состав архива DIM.

```
dim
dim.txt
dim_dcc
dim_demo
dim_docs
dim_manual.pdf
dim_overview.pdf
dim_test
dim\.setup
dim\bin
dim\dim
dim\DIM_Performance.pdf
dim\jdim
dim\LICENSE.GPL
dim\linux
dim\makefile
dim\makefile_benchmark
dim\makefile_common
dim\makefile_did
dim\makefile_dim
dim\makefile_examples
dim\makefile_util
dim\README.txt
dim\README_v10.txt
dim\README_v11.txt
dim\README_v12.txt
dim\README_v13.txt
dim\README_v14.txt
dim\README_v15.txt
dim\README_v16.txt
dim\README_v17.txt
dim\README_v18.txt
dim\README_v19.txt
dim\README_v20.txt
dim\README_v9.txt
dim\setup.sh
dim\src
dim\Visual
dim\bin\benchClient.exe
dim\bin\benchServer.exe
dim\bin\cpp_client.exe
dim\bin\cpp_server.exe
dim\bin\cpp_serverStd.exe
dim\bin\demo_client.exe
dim\bin\demo_server.exe
dim\bin\DID.exe
dim\bin\DID.INI
dim\bin\dim.dll
dim\bin\dim.exp
dim\bin\dim.lib
dim\bin\dimBridge.exe
dim\bin\dimStd.exp
dim\bin\dimStd.lib
dim\bin\DIMTree.exe
dim\bin\DIMTree.exe.pdf
dim\bin\DIMTree.ini
dim\bin\dim_get_service.exe
dim\bin\dim_send_command.exe
dim\bin\dns.exe
dim\bin\jdim.dll
dim\bin\jdim.exp
dim\bin\jdim.lib
dim\bin\Microsoft.VC80.CRT.manifest
dim\bin\msvcm80.dll
dim\bin\msvcp80.dll
dim\bin\msvcr80.dll
dim\bin\pvss_dim_client.exe
dim\bin\pvss_dim_server.exe
dim\bin\rpc_client.exe
dim\bin\rpc_server.exe
dim\bin\TestServer.exe
dim\bin\test_client.exe
dim\bin\test_server.exe
dim\dim\dic.h
dim\dim\dic.hxx
dim\dim\dim.h
```

```
dim\dim\dim.hxx
dim\dim\dim_common.h
dim\dim\dim_core.hxx
dim\dim\dim_jni.h
dim\dim\dim_tcpip.h
dim\dim\dis.h
dim\dim\dis.hxx
dim\dim\dllist.hxx
dim\dim\sllist.hxx
dim\dim\tokenstring.hxx
dim\jdim\classes
dim\jdim\jdimsources.txt
dim\jdim\makefile
dim\jdim\sources
dim\jdim\classes\dim
dim\jdim\classes\dim.jar
dim\jdim\classes\dim\.sniffdir
dim\jdim\classes\dim\Client$ReceiveSynchronizer.class
dim\jdim\classes\dim\Client$SendSynchronizer.class
dim\jdim\classes\dim\Client$Subscription.class
dim\jdim\classes\dim\Client.class
dim\jdim\classes\dim\CompletionHandler$ObjectInUse.class
dim\jdim\classes\dim\CompletionHandler.class
dim\jdim\classes\dim\DataDecoder.class
dim\jdim\classes\dim\DataEncoder.class
dim\jdim\classes\dim\DataOffset.class
dim\jdim\classes\dim\Dbg.class
dim\jdim\classes\dim\DimBrowser$1.class
dim\jdim\classes\dim\DimBrowser.class
dim\jdim\classes\dim\DimClient.class
dim\jdim\classes\dim\DimCommand.class
dim\jdim\classes\dim\DimCommandHandler.class
dim\jdim\classes\dim\DimCurrentInfo.class
dim\jdim\classes\dim\DimData.class
dim\jdim\classes\dim\DimDataOffsets.class
dim\jdim\classes\dim\DimErrorHandler$DimCltError.class
dim\jdim\classes\dim\DimErrorHandler$DimSrvError.class
dim\jdim\classes\dim\DimErrorHandler.class
dim\jdim\classes\dim\DimExitHandler$DimExit.class
dim\jdim\classes\dim\DimExitHandler.class
dim\jdim\classes\dim\DimInfo.class
dim\jdim\classes\dim\DimInfoHandler.class
dim\jdim\classes\dim\DimLock.class
dim\jdim\classes\dim\DimRpcInfo.class
dim\jdim\classes\dim\DimServer.class
dim\jdim\classes\dim\DimService.class
dim\jdim\classes\dim\DimServiceUpdateHandler.class
dim\jdim\classes\dim\DimTags.class
dim\jdim\classes\dim\DimTimer.class
dim\jdim\classes\dim\DimTimerHandler.class
dim\jdim\classes\dim\DimUpdatedInfo.class
dim\jdim\classes\dim\ErrorHandler.class
dim\jdim\classes\dim\ExitHandler.class
dim\jdim\classes\dim\Format.class
dim\jdim\classes\dim\IncorrectUsageException.class
dim\jdim\classes\dim\Memory$IllegalOffsetException.class
dim\jdim\classes\dim\Memory$ReadingOutOfBoundException.class
dim\jdim\classes\dim\Memory.class
dim\jdim\classes\dim\MultipleTaskCompletionSynchronizer.class
dim\jdim\classes\dim\MutableMemory$IllegalOffsetException.class
dim\jdim\classes\dim\MutableMemory$NoRoomException.class
dim\jdim\classes\dim\MutableMemory.class
dim\jdim\classes\dim\Native.class
dim\jdim\classes\dim\ObjectDescriptor.class
dim\jdim\classes\dim\Server$Command.class
dim\jdim\classes\dim\Server$Info.class
dim\jdim\classes\dim\Server$NameRedefined.class
dim\jdim\classes\dim\Server$NameUndefined.class
dim\jdim\classes\dim\Server$ReceiveSynchronizer.class
dim\jdim\classes\dim\Server.class
dim\jdim\classes\dim\SingleTaskCompletionSynchronizer.class
dim\jdim\classes\dim\Sizeof.class
dim\jdim\classes\dim\Tag.class
dim\jdim\classes\dim\test
dim\jdim\classes\dim\UtilityLib$StringSynchronizer.class
dim\jdim\classes\dim\UtilityLib.class
dim\jdim\classes\dim\.sniffdir\dim.proj.V0.42.sydat
dim\jdim\classes\dim\test\.sniffdir
dim\jdim\classes\dim\test\DimLock.class
dim\jdim\classes\dim\test\TestClient$1.class
dim\jdim\classes\dim\test\TestClient$2.class
```

```
dim\jdim\classes\dim\test\TestClient$3.class
dim\jdim\classes\dim\test\TestClient.class
dim\jdim\classes\dim\test\TestRPC.class
dim\jdim\classes\dim\test\TestServer$1.class
dim\jdim\classes\dim\test\TestServer$2.class
dim\jdim\classes\dim\test\TestServer$3.class
dim\jdim\classes\dim\test\TestServer.class
dim\jdim\classes\dim\test\TestSrvc.class
dim\jdim\classes\dim\test\TestSrvcMix.class
dim\jdim\classes\dim\test\TestSrvcStr.class
dim\jdim\classes\dim\test\TestUpdSrvc.class
dim\jdim\classes\dim\test\.sniffdir\test.proj.V0.42.sydat
dim\jdim\sources\dim
dim\jdim\sources\dim\.sniffdir
dim\jdim\sources\dim\Client.java
dim\jdim\sources\dim\CompletionHandler.java
dim\jdim\sources\dim\DataDecoder.java
dim\jdim\sources\dim\DataEncoder.java
dim\jdim\sources\dim\Dbg.java
dim\jdim\sources\dim\DimBrowser.java
dim\jdim\sources\dim\DimBrowser_old.java
dim\jdim\sources\dim\DimClient.java
dim\jdim\sources\dim\DimCommand.java
dim\jdim\sources\dim\DimCurrentInfo.java
dim\jdim\sources\dim\DimData.java
dim\jdim\sources\dim\DimDataOffsets.java
dim\jdim\sources\dim\DimErrorHandler.java
dim\jdim\sources\dim\DimExitHandler.java
dim\jdim\sources\dim\DimInfo.java
dim\jdim\sources\dim\DimLock.java
dim\jdim\sources\dim\DimRpcInfo.java
dim\jdim\sources\dim\DimServer.java
dim\jdim\sources\dim\DimService.java
dim\jdim\sources\dim\DimTags.java
dim\jdim\sources\dim\DimTimer.java
dim\jdim\sources\dim\DimUpdatedInfo.java
dim\jdim\sources\dim\Format.java
dim\jdim\sources\dim\IncorrectUsageException.java
dim\jdim\sources\dim\Memory.java
dim\jdim\sources\dim\MultipleTaskCompletionSynchronizer.java
dim\jdim\sources\dim\MutableMemory.java
dim\jdim\sources\dim\Native.java
dim\jdim\sources\dim\ObjectDescriptor.java
dim\jdim\sources\dim\Server.java
dim\jdim\sources\dim\SingleTaskCompletionSynchronizer.java
dim\jdim\sources\dim\Sizeof.java
dim\jdim\sources\dim\test
dim\jdim\sources\dim\UtilityLib.java
dim\jdim\sources\dim\.sniffdir\dim.proj.V0.42.sydat
dim\jdim\sources\dim\test\.sniffdir
dim\jdim\sources\dim\test\TestClient.java
dim\jdim\sources\dim\test\TestServer.java
dim\jdim\sources\dim\test\.sniffdir\test.proj.V0.42.sydat
dim\src\benchmark
dim\src\conn_handler.c
dim\src\copy_swap.c
dim\src\dic.c
dim\src\diccpp.cxx
dim\src\did
dim\src\dimcpp.cxx
dim\src\dim_jni.c
dim\src\dim_thr.c
dim\src\dim_thr_old.c
dim\src\dis.c
dim\src\discpp.cxx
dim\src\dis_old.c
dim\src\dll.c
dim\src\dna.c
dim\src\dns.c
dim\src\dtq.c
dim\src\examples
dim\src\feeserver.c
dim\src\hash.c
dim\src\open_dns.c
dim\src\sll.c
dim\src\swap.c
dim\src\tcpip.c
dim\src\tokenstring.cxx
dim\src\util
dim\src\utilities.c
dim\src\benchmark\benchClient.cxx
```

```
dim\src\benchmark\benchServer.cxx
dim\src\benchmark\bigClient.cxx
dim\src\benchmark\bigServer.cxx
dim\src\did\did.c
dim\src\did\did.h
dim\src\did\dui_colors.h
dim\src\did\dui_util.c
dim\src\did\dui_util.h
dim\src\examples\Copy of test_client.cxx
dim\src\examples\Copy of test_server.cxx
dim\src\examples\cpp_server.cxx
dim\src\examples\db_dim_client.c
dim\src\examples\db_dim_server.c
dim\src\examples\demo_client.c
dim\src\examples\demo_server.c
dim\src\examples\dim_fork.cxx
dim\src\examples\dim_fork2.cxx
dim\src\examples\Markus.cxx
dim\src\examples\pvss_dim_client.cxx
dim\src\examples\pvss_dim_server.cxx
dim\src\examples\rpc_client.cxx
dim\src\examples\rpc_server.cxx
dim\src\examples\rshServer.cxx
dim\src\examples\test_big_client.c
dim\src\examples\test_big_server.c
dim\src\examples\test_Browser.cxx
dim\src\examples\test_client.c
dim\src\examples\test_client.cxx
dim\src\examples\test_client1.c
dim\src\examples\test_client_ccpc.c
dim\src\examples\test_client_many.c
dim\src\examples\test_client_slac.c
dim\src\examples\test_serve.cxx
dim\src\examples\test_server.c
dim\src\examples\test_server.cxx
dim\src\examples\test_server.cxx_test
dim\src\examples\test_server1.c
dim\src\examples\test_serverFernando.cxx
dim\src\examples\test_server_ccpc.c
dim\src\examples\test_server_many.c
dim\src\examples\test_server_priorities.c
dim\src\examples\test_server_slac.c
dim\src\examples\test_tcp.c
dim\src\util\check_dim_servers.cxx
dim\src\util\check_dns.cxx
dim\src\util\dimbridge.cxx
dim\src\util\dim_get_service.c
dim\src\util\dim_send_command.c
dim\Visual\benchClient.dsp
dim\Visual\benchClient.vcproj
dim\Visual\benchServer.dsp
dim\Visual\benchServer.vcproj
dim\Visual\cpp_client.dsp
dim\Visual\cpp_client.vcproj
dim\Visual\cpp_server.dsp
dim\Visual\cpp_server.vcproj
dim\Visual\cpp_serverStd.dsp
dim\Visual\cpp_serverStd.vcproj
dim\Visual\demo_client.dsp
dim\Visual\demo_client.vcproj
dim\Visual\demo_server.dsp
dim\Visual\demo_server.vcproj
dim\Visual\dim.dsp
dim\Visual\dim.dsw
dim\Visual\dim.sln
dim\Visual\dim.vcproj
dim\Visual\dimBridge.dsp
dim\Visual\dimBridge.vcproj
dim\Visual\dimStd.dsp
dim\Visual\dimStd.vcproj
dim\Visual\dim_get_service.dsp
dim\Visual\dim_get_service.vcproj
dim\Visual\dim_send_command.dsp
dim\Visual\dim_send_command.vcproj
dim\Visual\dns.dsp
dim\Visual\dns.vcproj
dim\Visual\jdim.dsp
dim\Visual\jdim.vcproj
dim\Visual\pvss_dim_client.dsp
dim\Visual\pvss_dim_client.vcproj
dim\Visual\pvss_dim_server.dsp
```

```
dim\Visual\pvss_dim_server.vcproj
dim\Visual\rpc_client.dsp
dim\Visual\rpc_client.vcproj
dim\Visual\rpc_server.dsp
dim\Visual\rpc_server.vcproj
dim\Visual\test_client.dsp
dim\Visual\test_client.vcproj
dim\Visual\test_server.dsp
dim\Visual\test_server.vcproj
dim\Visual\test_tcp.vcproj
dim_dcc\_DIM.PAS
dim_demo\client.dpr
dim_demo\client.exe
dim_demo\demo.bat
dim_demo\make.bat
dim_demo\Readme.pdf
dim_demo\server.dpr
dim_demo\server.exe
dim_docs\dim_man_ru.odt
dim_docs\dim_man_ru.pdf
dim_docs\dim_pic_01.PNG
dim_docs\logo_cern.gif
dim_test\client.dpr
dim_test\client.exe
dim_test\make.bat
dim_test\server.dpr
dim_test\server.exe
dim_test\test.bat
```

Приложение 2. Модуль _DIM.PAS.

```
 {
 ******************************************************************************
 CRW32 project
 Copyright (C) by Kuryakin Alexey, Sarov, Russia, 2004, <kouriakine@mail.ru>
 DIM interface for Object Pascal (Delphi 5.0 tested).
 DIM is Distributed Information Manager by CERN, see http://dim.web.cern.ch/dim/
 DIM is CERN product under GNU GPL for distributed realtime DAQ & SCADA systems.
 DIM included in CRW32 "as is", I wrote only interface unit for Object Pascal.
 DIM is multiplatform: Windows, Linux, Unix etc.
 DNS is Distributed Name Server.
 DIS is a DIM server.
 DIC is a DIM client.
 Modifications:
 20040703 - creation & tests
 20040707 - first tested release
 20050616 - scheduler_class, priority etc
 20190213 - update to dim20; use TDimLong type (tags); add DIM/DIS/DIC stuff.
 ******************************************************************************
 }

unit _dim;                          // Distributed Information Manager interface

interface

 ////////////////////////////////////////
 // DIM general constants, derived from dim.h
 ////////////////////////////////////////
const
 DNS_TASK           = 'DIM_DNS';    // Name server task
 DNS_PORT           = 2505;         // Name server port, if DIM_DNS_PORT not defined
 START_PORT_RANGE   = 5100;         // Lowest  port to use by DIM servers
 STOP_PORT_RANGE    = 10000;        // Highest port to use by DIM servers


 ////////////////////////////////////////////
 // DIM service types, derived from dim_common.h
 ////////////////////////////////////////////
const
 ONCE_ONLY          = $01;          // Execute service once
 TIMED              = $02;          // Update service by timer
 MONITORED          = $04;          // Update service when changed (server take care)
 COMMAND            = $08;
 DIM_DELETE         = $10;
 MONIT_ONLY         = $20;
 UPDATE             = $40;
 TIMED_ONLY         = $80;
 MONIT_FIRST        = $100;
 MAX_TYPE_DEF       = $100;
 STAMPED            = $1000;

 ///////////////////////
 // DIM Error Severities
 ///////////////////////
const
 es_DIM_INFO        = 0;            // No error(s), just information
 es_DIM_WARNING     = 1;            // Warning, not error yet
 es_DIM_ERROR       = 2;            // Error found (recoverable)
 es_DIM_FATAL       = 3;            // Fatal error (unrecoverable)

 /////////////////
 // DIM Error Codes
 /////////////////
const
 ec_DIMDNSUNDEF     = $1;           // DIM_DNS_NODE undefined           FATAL
 ec_DIMDNSREFUS     = $2;           // DIM_DNS refuses connection       FATAL
 ec_DIMDNSDUPLC     = $3;           // Service already exists in DNS    FATAL
 ec_DIMDNSEXIT      = $4;           // DNS requests server to EXIT      FATAL
 ec_DIMDNSTMOUT     = $5;           // Server failed sending Watchdog   WARNING
 ec_DIMSVCDUPLC     = $10;          // Service already exists in Server ERROR
 ec_DIMSVCFORMT     = $11;          // Bad format string for service    ERROR
 ec_DIMSVCINVAL     = $12;          // Service ID invalid               ERROR
 ec_DIMSVCTOOLG     = $13;          // Service name too long            ERROR
 ec_DIMTCPRDERR     = $20;          // TCP/IP read error                ERROR
 ec_DIMTCPWRRTY     = $21;          // TCP/IP write   error - Retrying  WARNING
 ec_DIMTCPWRTMO     = $22;          // TCP/IP write error - Disconnect  ERROR
 ec_DIMTCPLNERR     = $23;          // TCP/IP listen error              ERROR
```

```
 ec_DIMTCPOPERR     = $24;          // TCP/IP open server error         ERROR
 ec_DIMTCPCNERR     = $25;          // TCP/IP connection error          ERROR
 ec_DIMTCPCNEST     = $26;          // TCP/IP connection established    INFO
 ec_DIMDNSCNERR     = $30;          // Connection to DNS failed         ERROR
 ec_DIMDNSCNEST     = $31;          // Connection to DNS established    INFO

type                                // dim_long is type uses to store tag values
 TDimLong           = Integer;      // => integer type with same size as Pointer

 ////////////////////////////////////////////////////////////////////
 // User callback routines to be called from DIM core to make all needs.
 // Note that all DIM routines uses 'cdecl' call convention.
 // TDis_User_Routine   : server callback to provide service data.
 // TDis_Cmnd_Routine   : server callback to execute command received from client.
 // TDis_Exit_Handler   : server callback to handle client or server exit.
 // TDic_User_Routine   : client callback to accept received service data.
 // TDic_Cmnd_Routine   : client callback to notify sent command result.
 // TDtq_User_Routine   : timeout callback to notify timeout expired.
 // TDis_Error_Handler  : server callback to handle errors.
 // TDic_Error_Handler  : client callback to handle errors.
 // TDim_Thread_Routine : callback routine for user thread.
 // tag                 : uses to identify service/command.
 // buff                : buffer to get/put service/command data.
 // size                : buffer size.
 // ret_code            : sent function result.
 // severity            : error class - info/warning/error/fatal
 // error_code          : error code to identify error
 // error_message       : error message to explain what's happened
 ////////////////////////////////////////////////////////////////////
type
 TDis_User_Routine   = procedure(var tag:TDimLong; var buff:Pointer; var size:Integer; var
first:Integer); cdecl;
 TDis_Cmnd_Routine   = procedure(var tag:TDimLong; buff:Pointer; var size:Integer); cdecl;
 TDis_Exit_Handler   = procedure(var code:Integer); cdecl;
 TDic_User_Routine   = procedure(var tag:TDimLong; buff:Pointer; var size:Integer); cdecl;
 TDic_Cmnd_Routine   = procedure(var tag:TDimLong; var ret_code:Integer); cdecl;
 TDtq_User_Routine   = procedure(tag:TDimLong); cdecl;
 TDis_Error_Handler  = procedure(severity:Integer; error_code:Integer; error_message:PChar); cdecl;
 TDic_Error_Handler  = procedure(severity:Integer; error_code:Integer; error_message:PChar); cdecl;
 TDim_Thread_Routine = procedure(tag:TDimLong); cdecl;

 ////////////////////////////////////////////////////////////////////////////
 // Note that all dim.dll routines uses 'cdecl' call convention.
 // Note that all dimStd.dll routines uses 'stdcall' call convention.
 // To use dimStd.dll instead of dim.dll, change all 'cdecl' to 'stdcall'.
 ////////////////////////////////////////////////////////////////////////////

 ////////////////////////////////////
 // DIM common API, derived from dim.h
 ////////////////////////////////////

 //
 // Hash function uses for hash table indexing.
 //
function  HashFunction(name:PChar; max:Integer):Integer; cdecl;

 //
 // Get host node name.
 // That is environment variable DIM_HOST_NODE or local host name.
 // Result : 1=Ok, 0=fail.
 //
function  get_node_name(NodeName:PChar):Integer; cdecl;

 //
 // Get DNS port number.
 // That is environment variable DIM_DNS_PORT or standard port DNS_PORT.
 //
function  get_dns_port_number:Integer; cdecl;

 //
 // Get DNS node name from environment.
 // That is environment variable DIM_DNS_NODE.
 // It's better to use dim_get_dns_node instead.
 // Result : 1=Ok, 0=fail.
 //
function  get_dns_node_name(NodeName:PChar):Integer; cdecl;

 //
 // Get list of DNS accepted domains from environment.
 // That is environment variable DIM_DNS_ACCEPTED_DOMAINS.
 // Result : 1=Ok, 0=fail.
```

```
 //
function  get_dns_accepted_domains(Domains:PChar):Integer; cdecl;

 //
 // Get list of DNS accepted nodes from environment.
 // That is environment variable DIM_DNS_ACCEPTED_NODES.
 // Result : 1=Ok, 0=fail.
 //
function get_dns_accepted_nodes(nodes:PChar):Integer; cdecl;

 ////////////////////////////////////////
 // DIM common API, derived from dim_common.h
 ////////////////////////////////////////

 //
 // Sleep until given time 'secs' (in seconds) elapsed.
 // Result : always 0.
 //
function  dtq_sleep(secs:Integer):Integer; cdecl;

 //
 // Start timeout request.
 // secs : timeout in seconds.
 // rout : user routine to be called when timeout interval elapsed.
 // tag  : uses to identify timeout request.
 //
procedure dtq_start_timer(secs:Integer; rout:TDtq_User_Routine; tag:TDimLong); cdecl;

 //
 // Stop (cancel) timeout request.
 // tag    : identifier of timeout request.
 // Result : time, in seconds, remains to timeout or -1 if timeout occured.
 //
function  dtq_stop_timer(tag:TDimLong):Integer; cdecl;

 //
 // Internal DIM routines.
 //
procedure dim_init; cdecl;
procedure dim_no_threads; cdecl;
procedure dna_set_test_write(conn_id:Integer; time:Integer); cdecl;
procedure dna_rem_test_write(conn_id:Integer); cdecl;

 //
 // dim_get/set_scheduler_class - get/set process priority class.
 // pclass   : -1/0/1/2=IDLE/NORMAL/HIGH/REALTIME
 // threadId : 1/2/3=Main/IO/Timer
 // prio     : -3/-2-1/0/1/2/3=IDLE/LOWEST/LOWER/NORMAL/HIGHER/HIGHEST/TIMECRITICAL
 // Returns 1/0=Ok/fail.
 //
function dim_set_scheduler_class(pclass:Integer):Integer; cdecl;
function dim_get_scheduler_class(var pclass:Integer):Integer; cdecl;
function dim_set_priority(threadId:Integer; prio:Integer):Integer; cdecl;
function dim_get_priority(threadId:Integer; var prio:Integer):Integer; cdecl;

 //
 // Get/set DIM_DNS_NODE, DIM_DNS_PORT. Buffer Node must be >= 40 chars. Returns 1/0=Ok/fail.
 //
function  dim_set_dns_node(Node:PChar):Integer; cdecl;
function  dim_get_dns_node(Node:PChar):Integer; cdecl;
function  dim_set_dns_port(port:Integer):Integer; cdecl;
function  dim_get_dns_port:Integer; cdecl;

 //
 // Internal DIM routines.
 //
procedure dic_set_debug_on; cdecl;
procedure dic_set_debug_off; cdecl;
procedure dim_print_date_time; cdecl;
procedure dim_set_write_timeout(secs:Integer); cdecl;
function  dim_get_write_timeout:Integer; cdecl;
procedure dim_usleep(t:Cardinal); cdecl;
function  dim_wait:Integer; cdecl;
procedure dim_pause; cdecl;
procedure dim_wake_up; cdecl;
procedure dim_lock; cdecl;
procedure dim_unlock; cdecl;
procedure dim_sleep(t:Cardinal); cdecl;
procedure dim_win_usleep(t:Cardinal); cdecl;
function  dim_start_thread(UserRoutine:TDim_Thread_Routine; tag:TDimLong):TDimLong; cdecl;
function  dic_set_dns_node(node:PChar):Integer; cdecl;
```

```
function  dic_get_dns_node(node:PChar):Integer; cdecl;
function  dic_set_dns_port(port:Integer):Integer; cdecl;
function  dic_get_dns_port:Integer; cdecl;
function  dis_set_dns_node(node:PChar):Integer; cdecl;
function  dis_get_dns_node(node:PChar):Integer; cdecl;
function  dis_set_dns_port(port:Integer):Integer; cdecl;
function  dis_get_dns_port:Integer; cdecl;
procedure dim_stop; cdecl;
function  dim_stop_thread(tid:TDimLong):Integer; cdecl;
function  dis_add_dns(node:PChar; port:Integer):TDimLong; cdecl;
function  dic_add_dns(node:PChar; port:Integer):TDimLong; cdecl;
function  dim_get_env_var(env_var:PChar; value:PChar; value_size:Integer):Integer; cdecl;
function  dim_set_write_buffer_size(bytes:Integer):Integer; cdecl;
function  dim_get_write_buffer_size:Integer; cdecl;
function  dim_set_read_buffer_size(bytes:Integer):Integer; cdecl;
function  dim_get_read_buffer_size:Integer; cdecl;
procedure dis_set_debug_on; cdecl;
procedure dis_set_debug_off; cdecl;
procedure dim_set_keepalive_timeout(secs:Integer); cdecl;
function  dim_get_keepalive_timeout:Integer; cdecl;
procedure dim_set_listen_backlog(size:Integer); cdecl;
function  dim_get_listen_backlog:Integer; cdecl;

 ////////////////////////////////////
 // DIM Server API, derived from dis.h
 ////////////////////////////////////

 //
 // Start serving client requests.
 // TaskName : server identifier, uses by client to access services.
 // Result   : 1 = Ok, 0 = fails.
 //
function  dis_start_serving(TaskName:PChar):Integer; cdecl;

 //
 // Stop serving, remove all DIM services.
 //
procedure dis_stop_serving; cdecl;

 //
 // Get from command queue next command requested by client.
 // Uses if UserRoutine=nil when dis_add_cmnd called.
 // Tag    : command identifier.
 // Buff   : buffer where command is to be copied to.
 // Size   : on input, buffer size; on output, received command size.
 // Result : 0 = no waiting commands, 1 = Ok, -1 = buffer size to small.
 //
function  dis_get_next_cmnd(var Tag:TDimLong; Buff:Pointer; var Size:Integer):Integer; cdecl;

 //
 // Get current client name as PID@hostname.
 // To be called inside TDis_User_Routine/TDis_Cmnd_Routine.
 // Name   :  buffer to copy client name to, at least 81 chars.
 // Result :  client connection ID or 0.
 //
function  dis_get_client(Name:PChar):Integer; cdecl;

 //
 // Get current client connection ID.
 // To be called inside TDis_User_Routine/TDis_Cmnd_Routine.
 // Result : Current connection ID.
 //
function  dis_get_conn_id:Integer; cdecl;

 //
 // Add new service to service list.
 // ServiceName : name to identify service.
 // ServiceType : describe service structure, like I:2;D:2;C.
 // ServiceAddr : points to service buffer or nil if uses UserRoutine.
 // ServiceSize : size of ServiceAddr buffer or 0 if uses UserRoutine.
 // UserRoutine : user routine to handle client requests.
 // Tag         : uses to identify service.
 // Returns     : Nonzero service ID if Ok or 0 if fails.
 //
function  dis_add_service(ServiceName:PChar; ServiceType:PChar;
                          ServiceAddr:Pointer; ServiceSize:Integer;
                          UserRoutine:TDis_User_Routine; Tag:TDimLong):Cardinal; cdecl;

 //
 // Add new command to server command list.
 // ServiceName : name to identify command.
```

```
 // ServiceType : describe command structure, like I:2;D:2;C.
 // UserRoutine : user routine to handle client commands.
 // Tag         : uses to identify service.
 // Returns     : Nonzero service ID if Ok or 0 if fails.
 //
function  dis_add_cmnd(ServiceName:PChar; ServiceType:PChar;
                       UserRoutine:TDis_Cmnd_Routine;
                       Tag:TDimLong):Cardinal; cdecl;
 //
 // Setup user routine to be called after client exit.
 // Client must declare Integer coomand service TASK/SET_EXIT_HANDLER
 // with some 'tag' value to identify client. UserRoutine will be called at
 // server side with 'tag' argument, when client exited. Client code like:
 // int id=123; sprintf(str,"%s/SET_EXIT_HANDLER",TaskName); dic_cmnd_service(str, &id, 4);
 //
procedure  dis_add_client_exit_handler(UserRoutine:TDis_Exit_Handler); cdecl;

 //
 // Set user routine to be called after client with given connection ID exit.
 // To get connection ID, use dis_get_conn_id, available inside UserRoutine.
 // Client may do nothing, code uses to identify reason, it should be <> 0.
 //
procedure  dis_set_client_exit_handler(ConnId:Integer; code:Integer); cdecl;

 //
 // Setup server user routine to be called when client call TASK/EXIT command.
 // For example, client wants to kill server. Server may die or not, as wish.
 //
procedure dis_add_exit_handler(UserRoutine:TDis_Exit_Handler); cdecl;

 //
 // Add an error handler to this server.
 //
procedure dis_add_error_handler(handler:TDis_Error_Handler); cdecl;

 //
 // ???
 //
procedure dis_report_service(ServiceName:PChar); cdecl;

 //
 // Update service with given ServiceID, send service data to notify clients.
 // ServiceID : service identifier returned by dis_add_service.
 // Result    : 1=Ok, 0=fails.
 //
function  dis_update_service(ServiceId:Cardinal):Integer; cdecl;

 //
 // Remove service with given ServiceID.
 // ServiceID : service identifier returned by dis_add_service.
 // Result    : 1=Ok, 0=fails.
 //
function  dis_remove_service(ServiceId:Cardinal):Integer; cdecl;

 //
 // Send service to clients. It's better to use dis_update_service instead.
 // ServiceID : service identifier returned by dis_add_service.
 // Buff,Size : Buffer where service placed and size of buffer.
 //
procedure dis_send_service(ServiceId:Cardinal; Buff:Pointer; Size:Integer); cdecl;

 //
 // Set packet size. Usually packet size grows dinamically, if packet size insufficient.
 // Size   : New packet size.
 // Result : 1=Ok, 0=fails.
 //
function  dis_set_buffer_size(Size:Integer):Integer; cdecl;
 //
 // Set quality for given ServiceId.
 //
procedure dis_set_quality(ServiceId:Cardinal; Quality:Integer); cdecl;

 //
 // Set time stamp for given ServiceId.
 //
procedure dis_set_timestamp(ServiceId:Cardinal; secs:Integer; millisecs:Integer); cdecl;

 //
 // Selective service update for given clients.
 // ServiceID      : service identifier returned by dis_add_service.
 // client_id_list : integer array of clients, terminated by 0.
```

```
 // Result         : 1=Ok, 0=fails.
 //
function  dis_selective_update_service(ServiceId:Cardinal; var client_id_list:Integer):Integer;
cdecl;

 //
 // Disable padding by data size.
 //
procedure dis_disable_padding; cdecl;

 //
 // Get timeout for given service & client.
 //
function  dis_get_timeout(ServiceId:Cardinal; ClientId:Integer):Integer; cdecl;

 //
 // Gets the names of DIM services that have an error.
 // To be called inside TDis_Error_Handler to find out, which service(s) have an error.
 // Returns a list of services, that originated the error.
 //
function dis_get_error_services:PChar; cdecl;

 //
 // Gets the services of a DIM client, which has subscribed to this DIM server.
 // To be called inside TDis_User_Routime when service data is updated.
 //
function dis_get_client_services(conn_id:Integer):PChar; cdecl;

 //
 // Functions for multiple DNS. Similar, but uses dns_id from dis_add_dns/dic_add_dns.
 //
function  dis_start_serving_dns(dns_id:TDimLong; task_name:PChar):Integer; cdecl;
procedure dis_stop_serving_dns(dns_id:TDimLong); cdecl;
function  dis_add_service_dns(dns_id:TDimLong; ServiceName:PChar; ServiceType:PChar;
                        ServiceAddr:Pointer; ServiceSize:Integer;
                        UserRoutine:TDis_User_Routine; Tag:TDimLong):Cardinal; cdecl;
function  dis_add_cmnd_dns(dns_id:TDimLong; ServiceName:PChar; ServiceType:PChar;
                      UserRoutine:TDis_Cmnd_Routine; Tag:TDimLong):Cardinal; cdecl;

 //
 // Get number of clients connected to the service.
 //
function dis_get_n_clients(ServiceId:Cardinal):Integer; cdecl;

 //
 // Get time stamp of a service.
 //
function dis_get_timestamp(ServiceId:Cardinal; var secs:Integer; var millisecs:Integer):Integer;
cdecl;

 ////////////////////////////////////
 // DIM client API, derived from dic.h
 ////////////////////////////////////

 //
 // Start information service of different kind (stamped, unstamped, command,
 // command with callback notification), provided by server.
 // ServiceName : name of service.
 // req_type    : service tipe, ONCE_ONLY,TIMED,MONITORED
 // req_timeout : timeout in seconds.
 // ServiceAddr : Service buffer address or nil if uses UserRoutine.
 // ServiceSize : Service buffer size    or 0   if uses UserRoutine.
 // UserRoutine : User routine to handle service data or nil if uses ServiceAddr.
 // CmndRoutine : User routine to notify if command was sent properly or not.
 // tag         : service identifier.
 // fill_addr   : buffer returned if unsuccess IO.
 // fill_size   : previous buffer size.
 // Result      : ServiceID or 0 if service unaccessible.
 //
function  dic_info_service(ServiceName:PChar; req_type:Integer; req_timeout:Integer;
                         ServiceAddr:Pointer; ServiceSize:Integer;
                         UserRoutine:TDic_User_Routine; tag:TDimLong;
                         fill_addr:Pointer; fill_size:Integer):Cardinal; cdecl;
function  dic_info_service_stamped(ServiceName:PChar; req_type:Integer; req_timeout:Integer;
                                 ServiceAddr:Pointer; ServiceSize:Integer;
                                 UserRoutine:TDic_User_Routine; tag:TDimLong;
                                 fill_addr:Pointer; fill_size:Integer):Cardinal; cdecl;
function  dic_cmnd_callback(ServiceName:PChar; ServiceAddr:Pointer; ServiceSize:Integer;
                          CmndRoutine:TDic_Cmnd_Routine; tag:TDimLong):Integer; cdecl;
function  dic_cmnd_service(ServiceName:PChar; ServiceAddr:Pointer; ServiceSize:Integer):Integer;
cdecl;
```

```
 //
 // Change service buffer address and size.
 //
procedure dic_change_address(ServiceId:Cardinal; ServiceAddr:Pointer; ServiceSize:Integer); cdecl;

 //
 // Release service given by ServiceID.
 //
procedure dic_release_service(ServiceId:Cardinal); cdecl;

 //
 // Get client ID in pid@node form, like 123@lxplus.cern.ch.
 // Name   : buffer to write client ID.
 // Result : always 1.
 //
function  dic_get_id(Name:PChar):Integer; cdecl;

 //
 // Get quality for given ServiceId.
 // If ServiceId=0, get quality for current service (uses in user routines).
 // Result : quality or -1 if invalid service.
 //
function  dic_get_quality(ServiceId:Cardinal):Integer; cdecl;

 //
 // Get time stamp for given ServiceId.
 // Result : -1=fails, 0=no-stamped service, 1=Ok.
 //
function  dic_get_timestamp(ServiceId:Cardinal; var secs:Integer; var milisecs:Integer):Integer;
cdecl;

 //
 // Get format for given ServiceId.
 // If ServiceId=0, get format for current service (uses in user routines).
 // Result : format char or nil if invalid service.
 //
function  dic_get_format(ServiceId:Cardinal):PChar; cdecl;

 //
 // Disable data size padding.
 //
procedure dic_disable_padding; cdecl;

 //
 // Close DNS connection.
 //
procedure  dic_close_dns; cdecl;

 //
 // Add an error handler to this client.
 //
procedure dic_add_error_handler(handler:TDic_Error_Handler); cdecl;

 //
 // Returns a list of services, that originated the error.
 // To be called inside TDic_Error_Handler to find out, which service(s) have an error.
 //
function dic_get_error_services:PChar; cdecl;

 //
 // Gets the services of a DIM server to which the client has subscribed.
 // To be called inside TDic_User_Routine when service data is being received from a server.
 // Its purpose is to get a list of all services, to which this DIM client has subscribed.
 // The list contains only the services of the specified server.
 //
function dic_get_server_services(conn_id:Integer):PChar; cdecl;

 //
 // Gets the identification of the current DIM server.
 // To be called inside TDic_User_Routine when service data is being received from a server.
 // Returns the server_id, in terms of its connection id, or 0 if no server is currently being
handled.
 // It obtains the name of the DIM server in the format "DIMSERVERNAME@nodename".
 //
function dic_get_server(name:PChar):Integer; cdecl;

 //
 // Gets the connection ID of the current DIM server.
 // To be called inside TDic_User_Routine when service data is being received from a server.
 // Its purpose is to obtain the conn_id that is required by other routines.
```

```
 //
function dic_get_conn_id:Integer; cdecl;

 //
 // Undocumented.
 // Stop DIM client connections and threads.
 //
procedure dic_stop; cdecl;

 //
 // Undocumented.
 // To be called inside TDic_User_Routine when service data is being received from a server.
 // Returns the server_id, in terms of its connection id, or 0 if no server is currently being
handled.
 // It obtains the PID of the DIM server.
 //
function dic_get_server_pid(var pid:Integer):Integer; cdecl;

 ////////////////////////
 // Addon utility routines
 ////////////////////////

 //
 // Name of severity: INFO/WARNING/ERROR/FATAL/UNKNOWN
 //
function dim_severity_name(severity:Integer):PChar; cdecl;

implementation

const
 DimDll = 'dim.dll';

function  HashFunction;                external DimDll name 'HashFunction';
function  get_node_name;               external DimDll name 'get_node_name';
function  get_dns_port_number;         external DimDll name 'get_dns_port_number';
function  get_dns_node_name;           external DimDll name 'get_dns_node_name';
function  get_dns_accepted_domains;    external DimDll name 'get_dns_accepted_domains';
function  get_dns_accepted_nodes;      external DimDll name 'get_dns_accepted_nodes';
function  dtq_sleep;                   external DimDll name 'dtq_sleep';
procedure dtq_start_timer;             external DimDll name 'dtq_start_timer';
function  dtq_stop_timer;              external DimDll name 'dtq_stop_timer';
procedure dim_init;                    external DimDll name 'dim_init';
procedure dim_no_threads;              external DimDll name 'dim_no_threads';
procedure dna_set_test_write;          external DimDll name 'dna_set_test_write';
procedure dna_rem_test_write;          external DimDll name 'dna_rem_test_write';
function  dim_set_scheduler_class;     external DimDll name 'dim_set_scheduler_class';
function  dim_get_scheduler_class;     external DimDll name 'dim_get_scheduler_class';
function  dim_set_priority;            external DimDll name 'dim_set_priority';
function  dim_get_priority;            external DimDll name 'dim_get_priority';
function  dim_set_dns_node;            external DimDll name 'dim_set_dns_node';
function  dim_get_dns_node;            external DimDll name 'dim_get_dns_node';
function  dim_set_dns_port;            external DimDll name 'dim_set_dns_port';
function  dim_get_dns_port;            external DimDll name 'dim_get_dns_port';
procedure dic_set_debug_on;            external DimDll name 'dic_set_debug_on';
procedure dic_set_debug_off;           external DimDll name 'dic_set_debug_off';
procedure dim_print_date_time;         external DimDll name 'dim_print_date_time';
procedure dim_set_write_timeout;       external DimDll name 'dim_set_write_timeout';
function  dim_get_write_timeout;       external DimDll name 'dim_get_write_timeout';
procedure dim_usleep;                  external DimDll name 'dim_usleep';
function  dim_wait;                    external DimDll name 'dim_wait';
procedure dim_pause;                   external DimDll name 'dim_pause';
procedure dim_wake_up;                 external DimDll name 'dim_wake_up';
procedure dim_lock;                    external DimDll name 'dim_lock';
procedure dim_unlock;                  external DimDll name 'dim_unlock';
procedure dim_sleep;                   external DimDll name 'dim_sleep';
procedure dim_win_usleep;              external DimDll name 'dim_win_usleep';
function  dim_start_thread;            external DimDll name 'dim_start_thread';
function  dic_set_dns_node;            external DimDll name 'dic_set_dns_node';
function  dic_get_dns_node;            external DimDll name 'dic_get_dns_node';
function  dic_set_dns_port;            external DimDll name 'dic_set_dns_port';
function  dic_get_dns_port;            external DimDll name 'dic_get_dns_port';
function  dis_set_dns_node;            external DimDll name 'dis_set_dns_node';
function  dis_get_dns_node;            external DimDll name 'dis_get_dns_node';
function  dis_set_dns_port;            external DimDll name 'dis_set_dns_port';
function  dis_get_dns_port;            external DimDll name 'dis_get_dns_port';
procedure dim_stop;                    external DimDll name 'dim_stop';
function  dim_stop_thread;             external DimDll name 'dim_stop_thread';
function  dis_add_dns;                 external DimDll name 'dis_add_dns';
function  dic_add_dns;                 external DimDll name 'dic_add_dns';
function  dim_get_env_var;             external DimDll name 'dim_get_env_var';
function  dim_set_write_buffer_size;   external DimDll name 'dim_set_write_buffer_size';
```

```
function  dim_get_write_buffer_size;    external DimDll name 'dim_get_write_buffer_size';
function  dim_set_read_buffer_size;     external DimDll name 'dim_set_read_buffer_size';
function  dim_get_read_buffer_size;     external DimDll name 'dim_get_read_buffer_size';
procedure dis_set_debug_on;             external DimDll name 'dis_set_debug_on';
procedure dis_set_debug_off;            external DimDll name 'dis_set_debug_off';
procedure dim_set_keepalive_timeout;    external DimDll name 'dim_set_keepalive_timeout';
function  dim_get_keepalive_timeout;    external DimDll name 'dim_get_keepalive_timeout';
procedure dim_set_listen_backlog;       external DimDll name 'dim_set_listen_backlog';
function  dim_get_listen_backlog;       external DimDll name 'dim_get_listen_backlog';
function  dis_start_serving;            external DimDll name 'dis_start_serving_';
procedure dis_stop_serving;             external DimDll name 'dis_stop_serving_';
function  dis_get_next_cmnd;            external DimDll name 'dis_get_next_cmnd_';
function  dis_get_client;               external DimDll name 'dis_get_client_';
function  dis_get_conn_id;              external DimDll name 'dis_get_conn_id';
function  dis_add_service;              external DimDll name 'dis_add_service_';
function  dis_add_cmnd;                 external DimDll name 'dis_add_cmnd_';
procedure dis_add_client_exit_handler;  external DimDll name 'dis_add_client_exit_handler_';
procedure dis_set_client_exit_handler;  external DimDll name 'dis_set_client_exit_handler_';
procedure dis_add_exit_handler;         external DimDll name 'dis_add_exit_handler_';
procedure dis_add_error_handler;        external DimDll name 'dis_add_error_handler';
procedure dis_report_service;           external DimDll name 'dis_report_service_';
function  dis_update_service;           external DimDll name 'dis_update_service_';
function  dis_remove_service;           external DimDll name 'dis_remove_service_';
procedure dis_send_service;             external DimDll name 'dis_send_service_';
function  dis_set_buffer_size;          external DimDll name 'dis_set_buffer_size';
procedure dis_set_quality;              external DimDll name 'dis_set_quality_';
procedure dis_set_timestamp;            external DimDll name 'dis_set_timestamp_';
function  dis_selective_update_service; external DimDll name 'dis_selective_update_service_';
procedure dis_disable_padding;          external DimDll name 'dis_disable_padding';
function  dis_get_timeout;              external DimDll name 'dis_get_timeout';
function  dis_get_error_services;       external DimDll name 'dis_get_error_services';
function  dis_get_client_services;      external DimDll name 'dis_get_client_services';
function  dis_start_serving_dns;        external DimDll name 'dis_start_serving_dns';
procedure dis_stop_serving_dns;         external DimDll name 'dis_stop_serving_dns';
function  dis_add_service_dns;          external DimDll name 'dis_add_service_dns';
function  dis_add_cmnd_dns;             external DimDll name 'dis_add_cmnd_dns';
function  dis_get_n_clients;            external DimDll name 'dis_get_n_clients';
function  dis_get_timestamp;            external DimDll name 'dis_get_timestamp_';
function  dic_info_service;             external DimDll name 'dic_info_service_';
function  dic_info_service_stamped;     external DimDll name 'dic_info_service_stamped_';
function  dic_cmnd_callback;            external DimDll name 'dic_cmnd_callback_';
function  dic_cmnd_service;             external DimDll name 'dic_cmnd_service_';
procedure dic_change_address;           external DimDll name 'dic_change_address_';
procedure dic_release_service;          external DimDll name 'dic_release_service_';
function  dic_get_id;                   external DimDll name 'dic_get_id_';
function  dic_get_quality;              external DimDll name 'dic_get_quality_';
function  dic_get_timestamp;            external DimDll name 'dic_get_timestamp_';
function  dic_get_format;               external DimDll name 'dic_get_format_';
procedure dic_disable_padding;          external DimDll name 'dic_disable_padding';
procedure dic_close_dns;                external DimDll name 'dic_close_dns';
procedure dic_add_error_handler;        external DimDll name 'dic_add_error_handler';
function  dic_get_error_services;       external DimDll name 'dic_get_error_services';
function  dic_get_server_services;      external DimDll name 'dic_get_server_services';
function  dic_get_server;               external DimDll name 'dic_get_server';
function  dic_get_conn_id;              external DimDll name 'dic_get_conn_id';
procedure dic_stop;                     external DimDll name 'dic_stop';
function  dic_get_server_pid;           external DimDll name 'dic_get_server_pid';

function dim_severity_name(severity:Integer):PChar; cdecl;
begin
 case severity of
  es_DIM_INFO    : Result:='INFO';
  es_DIM_WARNING : Result:='WARNING';
  es_DIM_ERROR   : Result:='ERROR';
  es_DIM_FATAL   : Result:='FATAL';
  else             Result:='UNKNOWN';
 end;
end;

end.
```

Приложение 3. Программа SERVER.DPR.

```
program server;                                 // Demo server to illustrate DIM features.

{$APPTYPE CONSOLE}

uses Windows, SysUtils, _dim;

var                                             // DIM uses data:
 dns_version  : Integer = 0;                    // To receive DNS version
 no_link      : Integer = -1;                   // To mark "server die" event.
 InfoServices : array of Integer = nil;         // List of published inform. DIM services
 CmndServices : array of Integer = nil;         // List of published command DIM services
 Terminated   : Boolean = false;                // Terminator for main loop
 Info1        : record                          // User data set #1
  CallCount   : Integer;                        // Call counter
  TickCount   : Integer;                        // GetTickCount
  Now         : Double;                         // Current time
 end;
 Info2        : record                          // User data set #2
  FileTime    : TFileTime;                      // System time as file time
  DateTime    : array[byte] of char;            // Date and time string
 end;

 //
 // Thread safe printing: Print(s,1) => StdOut, Print(s,2) => StdErr
 //
function Print(const Msg:AnsiString; n:Integer=1):DWORD;
var h:THandle;
begin
 if n=1 then h:=GetStdHandle(STD_OUTPUT_HANDLE) else
 if n=2 then h:=GetStdHandle(STD_ERROR_HANDLE) else h:=0;
 if not WriteFile(h,PChar(Msg)^,Length(Msg),Result,nil) then Result:=0;
end;

function PrintLn(const Msg:AnsiString; n:Integer=1):DWORD;
const Delimeter=#13#10;
begin
 Result:=Print(Msg+Delimeter,n);
end;

 //
 // Callback to receive DNS version
 //
procedure on_got_dns_version(var tag:TDimLong; buff:Pointer; var size:Integer); cdecl;
begin
 try
  if (size=SizeOf(no_link)) and (Integer(buff^) = no_link) then begin
   PrintLn('DNS server is dead. Please restart DNS.EXE.',2);
   Exit;
  end;
  if size=SizeOf(dns_version) then begin
   PrintLn(Format('Got DNS version: %d, tag:%d',[Integer(buff^),tag]));
   dns_version:=Integer(buff^);
   Exit;
  end;
  PrintLn(Format('Invalid data size: %d, tag:%d',[size,tag]),2);
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

 //
 // Callback on server exit
 //
procedure on_server_exit(var code:Integer); cdecl;
var name:array[byte] of char;
begin
 try
  if dis_get_client(name)=0 then name:='?';
  PrintLn(Format('Server "%s" (code %d) exited.',[name,code]));
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

 //
 // Callback on client exit
```

```
 //
procedure on_client_exit(var code:Integer); cdecl;
var name:array[byte] of char;
begin
 try
  if dis_get_client(name)=0 then name:='?';
  PrintLn(Format('Client "%s" disconnected from service %d.',[name,code]));
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

 //
 // Callback on server update - provide published information services
 //
procedure on_serve(var tag:TDimLong; var buff:Pointer; var size:Integer; var first:Integer); cdecl;
var name:array[byte] of char; i:Integer;
begin
 size:=0;
 buff:=nil;
 try
  if dis_get_client(name)=0 then name:='?';
  PrintLn(Format('Serving: service %d, client "%s"',[tag,name]));
  case tag of
   1: begin // Serve dataset #1 : Update content, assign buff & size
       Inc(Info1.CallCount); Info1.TickCount:=GetTickCount; Info1.Now:=Now;
       PrintLn(Format('Provide data: %d, %d, %g',[Info1.CallCount,Info1.TickCount,Info1.Now]));
       size:=SizeOf(Info1);
       buff:=@Info1;
      end;
   2: begin // Serve dataset #2 : Update content, assign buff & size
       GetSystemTimeAsFileTime(Info2.FileTime); StrPCopy(Info2.DateTime,DateTimeToStr(Now));
       PrintLn(Format('Provide data: %d, %s',[Int64(Info2.FileTime),Info2.DateTime]));
       size:=SizeOf(Info2.FileTime)+StrLen(Info2.DateTime)+1;
       buff:=@Info2;
      end;
   else PrintLn(Format('Unknown service identifier %d requested.',[tag]));
  end;
  if first>0 then begin
   if dis_get_client(name)=0 then name:='?';
   PrintLn(Format('Client "%s" connected to service %d.',[name,tag]));
   dis_set_client_exit_handler(dis_get_conn_id,tag);
   if (dis_get_client_services(dis_get_conn_id)<>nil)
   then PrintLn('Client services:
'+StringReplace(Trim(dis_get_client_services(dis_get_conn_id)),#10,' ',[rfReplaceAll]));
   Print('Client counter(s):');
   for i:=Low(InfoServices) to High(InfoServices) do
   Print(' '+IntToStr(dis_get_n_clients(InfoServices[i])));
   PrintLn('');
  end;
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

 //
 // Callback on client command - handle it
 //
procedure on_command(var tag:TDimLong; cmnd:Pointer; var size:Integer); cdecl;
begin
 try
  PrintLn(Format('Got command: %s, tag: %d',[PChar(cmnd),tag]));
  if(StrIComp(cmnd,'RESET')=0) then Info1.CallCount:=0;
  if(StrIComp(cmnd,'EXIT')=0) then Terminated:=true;
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

 //
 // Callback on Client Error
 //
procedure on_client_error(severity:Integer; error_code:Integer; error_message:PChar); cdecl;
begin
 try
  PrintLn(Format('%s - %s (%d) - %s',[FormatDateTime('yyyy.mm.dd-hh:nn:ss',Now),
                                      dim_severity_name(severity),error_code,error_message]));
  if (dic_get_error_services<>nil) and (strlen(dic_get_error_services)>0)
  then PrintLn('Error services: '+dic_get_error_services);
 except
  on E:Exception do PrintLn(E.Message,2);
```

```
 end;
end;

 //
 // Callback on Server Error
 //
procedure on_server_error(severity:Integer; error_code:Integer; error_message:PChar); cdecl;
begin
 try
  PrintLn(Format('%s - %s (%d) - %s',[FormatDateTime('yyyy.mm.dd-hh:nn:ss',Now),
                                      dim_severity_name(severity),error_code,error_message]));
  if (dis_get_error_services<>nil) and (strlen(dis_get_error_services)>0)
  then PrintLn('Error services: '+dis_get_error_services);
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

procedure Main;
var i:Integer; dns_node:array[byte] of char;
begin
 try
  //
  // 1. Initialize DIM, set DNS node & exit handlers.
  //
  dim_init;
  dis_disable_padding;
  dic_disable_padding;
  dic_add_error_handler(on_client_error);
  dis_add_error_handler(on_server_error);
  if dim_get_dns_node(dns_node)=0 then raise Exception.Create('Fail get DNS.');
  if StrLen(dns_node)=0 then dim_set_dns_node('localhost');
  if dim_get_dns_node(dns_node)=0 then raise Exception.Create('Fail get DNS.');
  if StrLen(dns_node)=0 then raise Exception.Create('Fail set DNS.');
  PrintLn('DIM_DNS_NODE='+dns_node);
  dis_add_client_exit_handler(on_client_exit);
  dis_add_exit_handler(on_server_exit);
  //
  // 2. Subscribe DNS server version to call it only once.
  //
  dic_info_service('DIS_DNS/VERSION_NUMBER',    // DNS server version service name
                   ONCE_ONLY,                   // Call it only once
                   10,                          // Timeout, seconds
                   nil, 0, on_got_dns_version,  // Callback to receive DNS version
                   1,                           // Tag to identify this service
                   @no_link, SizeOf(no_link)    // Data to send in case of timeout
                   );
  //
  // 3. Publish information services.
  //
  SetLength(InfoServices,2);
  InfoServices[0]:=dis_add_service('DEMO/INFO1', // Name of information service
                                   'I:2;D',      // Data format: two integers and double
                                   nil, 0,       // No constant buffer, use callback
                                   on_serve,     // Callback to provide service data
                                   1);           // Tag to identify this service
  InfoServices[1]:=dis_add_service('DEMO/INFO2', // Name of information service
                                   'I:2;C',      // Data format: two integers and chars
                                   nil, 0,       // No constant buffer, use callback
                                   on_serve,     // Callback to provide service data
                                    2);          // Tag to identify this service
  //
  // 4.Publish command services.
  //
  SetLength(CmndServices,2);
  CmndServices[0]:=dis_add_cmnd('DEMO/COMMAND1',  // Name of command service
                                'C',              // Data format
                                on_command,       // Callback to handle the command
                                1);               // Tag to identify this service
  CmndServices[2]:=dis_add_cmnd('DEMO/COMMAND2',  // Name of command service
                                'C',              // Data format
                                on_command,       // Callback to handle the command
                                2);               // Tag to identify this service
  //
  // 5. Start to serve published services.
  //
  if dis_start_serving('DEMO_TASK')=0 then begin
   PrintLn('Could not start serving!');
   Sleep(2000);
   Exit;
  end;
```

```
  //
  // 6. Main server loop:
  //    Update published information services each second
  //
  Info1.CallCount:=0;
  PrintLn('Start serving...');
  while not Terminated do begin
   for i:=0 to Length(InfoServices)-1 do
   dis_update_service(InfoServices[i]);
   Sleep(1000);
  end;
  //
  // 7. Stop DIM.
  //
  PrintLn('Stop DIM...');
  for i:=0 to Length(InfoServices)-1 do dis_remove_service(InfoServices[i]);
  for i:=0 to Length(CmndServices)-1 do dis_remove_service(CmndServices[i]);
  dis_stop_serving; dim_stop;
  PrintLn('Goodbye.');
  Sleep(2000);
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

begin
 Main;
end.
```

Приложение 4. Программа CLIENT.DPR.

```
program client;                              // Test client to illustrate DIM features.

{$APPTYPE CONSOLE}

uses
  Windows, SysUtils, _dim;

var
 dns_version  : Integer = 0;                 // To receive DNS version
 no_link      : Integer = -1;                // To mark "server die" event.
 InfoServices : array of Integer = nil;      // List of subscribed inform. DIM services
 Terminated   : Boolean = false;             // Terminator for main loop
 Info1        : record                       // 1 user published data set
  CallCount   : Integer;                     // Call counter
  TickCount   : Integer;                     // GetTickCount
  Now         : Double;                      // Current time
 end;
 Info2        : record                       // 2 user published data set
  FileTime    : TFileTime;                   // System time as file time
  DateTime    : array[byte] of char;         // Date and time string
 end;
 cmnd         : array[byte] of char;         // Command to send to server

 //
 // Thread safe printing: Print(s,1) => StdOut, Print(s,2) => StdErr
 //
function Print(const Msg:AnsiString; n:Integer=1):DWORD;
var h:THandle;
begin
 if n=1 then h:=GetStdHandle(STD_OUTPUT_HANDLE) else
 if n=2 then h:=GetStdHandle(STD_ERROR_HANDLE) else h:=0;
 if not WriteFile(h,PChar(Msg)^,Length(Msg),Result,nil) then Result:=0;
end;

function PrintLn(const Msg:AnsiString; n:Integer=1):DWORD;
const Delimeter=#13#10;
begin
 Result:=Print(Msg+Delimeter,n);
end;

 //
 // Callback on receive DNS version
 //
procedure on_got_dns_version(var tag:TDimLong; buff:Pointer; var size:Integer); cdecl;
begin
 try
  if (size=SizeOf(no_link)) and (Integer(buff^) = no_link) then begin
   PrintLn('DNS server is dead. Please restart DNS.EXE.');
   Exit;
  end;
  if size=SizeOf(dns_version) then begin
   PrintLn(Format('Got DNS version: %d, tag:%d',[Integer(buff^),tag]));
   dns_version:=Integer(buff^);
   Exit;
  end;
  PrintLn(Format('Invalid data size: %d, tag:%d',[size,tag]));
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

 //
 // Callback on receive information service data
 //
procedure on_got_data(var tag:TDimLong; buff:Pointer; var size:Integer); cdecl;
var node:array[byte] of char; pid:Integer;
begin
 try
  if (size=SizeOf(no_link)) and (Integer(buff^) = no_link) then begin
   PrintLn(Format('Service %d disconnected. Please restart server.',[tag]));
   Exit;
  end;
  node:=''; pid:=0; // Get server info
  if dic_get_server(node)=0 then node:='';
  if dic_get_server_pid(pid)=0 then pid:=0;
  case tag of
```

```
   1: begin // Receive dataset #1
       if size=SizeOf(Info1) then begin
        Move(buff^,Info1,size);
        PrintLn(Format('Got data: CallCount=%d, TickCount=%d, Now=%g from server %s PID %d',
                    [Info1.CallCount,Info1.TickCount,Info1.Now,node,pid]));
       end else PrintLn(Format('Invalid data set %d',[tag]));
      end;
   2: begin // Receive dataset #2
       if size<=SizeOf(Info2) then begin
        Move(buff^,Info2,size);
        PrintLn(Format('Got data: FileTime=%d, DateTime=%s from server %s PID %d',
                    [Int64(Info2.FileTime),Info2.DateTime,node,pid]));
       end else PrintLn(Format('Invalid data set %d',[tag]));
      end;
   else PrintLn(Format('Invalid service %d',[tag]));
  end;
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

 //
 // Callback on command sent to server
 //
procedure on_cmnd_sent(var tag:TDimLong; var ret_code:Integer); cdecl;
begin
 try
  if ret_code>0
  then PrintLn(Format('Sent command %d - Ok',[tag]))
  else PrintLn(Format('Could not send command %d',[tag]));
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

 //
 // Callback on Client Error
 //
procedure on_client_error(severity:Integer; error_code:Integer; error_message:PChar); cdecl;
begin
 try
  PrintLn(Format('%s - %s (%d) - %s',[FormatDateTime('yyyy.mm.dd-hh:nn:ss',Now),
                                      dim_severity_name(severity),error_code,error_message]));
  if (dic_get_error_services<>nil) and (strlen(dic_get_error_services)>0)
  then PrintLn('Error services: '+dic_get_error_services);
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

procedure Main;
var i:Integer; dns_node:array[byte] of char;
begin
 try
  //
  // 1. Initialize DIM, set DNS node.
  //
  dim_init;
  dis_disable_padding;
  dic_disable_padding;
  dic_add_error_handler(on_client_error);
  if dim_get_dns_node(dns_node)=0 then raise Exception.Create('Fail get DNS.');
  if StrLen(dns_node)=0 then dim_set_dns_node('localhost');
  if dim_get_dns_node(dns_node)=0 then raise Exception.Create('Fail get DNS.');
  if StrLen(dns_node)=0 then raise Exception.Create('Fail set DNS.');
  PrintLn('DIM_DNS_NODE='+dns_node);
  //
  // 2. Subscribe DNS server version to call it only once.
  //
  dic_info_service('DIS_DNS/VERSION_NUMBER',        // DNS server version service name
                   ONCE_ONLY,                       // Call it only once
                   10,                              // Timeout, seconds
                   nil, 0, on_got_dns_version,      // Callback to receive DNS version
                   1,                               // Tag to identify this service
                   @no_link, SizeOf(no_link));      // Data to send in case of timeout
  //
  // 3. Subscribe information services published by server.
  //
  SetLength(InfoServices,2);
  InfoServices[0]:=dic_info_service('DEMO/INFO1',   // Name of published service
                   MONITORED,                       // Refresh by server only
```

```
                    10,                                 // Timeout, sec, on server die
                    nil, 0, on_got_data,                // Callback on data received
                    1,                                  // Tag to identify this service
                    @no_link, SizeOf(no_link));         // Data to send in case of timeout
  InfoServices[1]:=dic_info_service('DEMO/INFO2',       // Name of published service
                    MONITORED,                          // Refresh by server only
                    10,                                 // Timeout, sec, on server die
                    nil, 0, on_got_data,                // Callback on data received
                    2,                                  // Tag to identify this service
                    @no_link, SizeOf(no_link));         // Data to send in case of timeout
  //
  // 4. Main client loop
  //
  while not Terminated do begin
   PrintLn('Enter command to send to server (RESET for example):');
   readln(cmnd); if strlen(cmnd)=0 then continue;
   if StrIComp(cmnd,'QUIT')=0 then break;
   PrintLn(Format('Sending command: "%s"',[cmnd]));
   dic_cmnd_callback('DEMO/COMMAND1',                   // Name of command service
                     @cmnd,strlen(cmnd)+1,              // Command to send
                     on_cmnd_sent,                      // Callback on data sent
                     1);                                // Tag to identify this service
  end;
  //
  // 5. Stop DIM.
  //
  PrintLn('Stop DIM...');
  for i:=0 to Length(InfoServices)-1 do dic_release_service(InfoServices[i]);
  dic_close_dns; dic_stop;
  PrintLn('Goodbye.');
  Sleep(2000);
 except
  on E:Exception do PrintLn(E.Message,2);
 end;
end;

begin
 Main;
end.
```

Приложение 5. Скрипт DEMO.BAT.

```
@echo off
del /f /q *.~dpr *.~dsk *.~pas *.dcu client.cfg server.cfg client.dsk server.dsk client.dof
server.dof 1>nul 2>nul

set dim_dns_node=%ComputerName%
set dim_dns_node=localhost

for %%i in ( "%~dp0..\dim\bin" ) do path %%~fi;%path%

set tasklist=..\dim\bin\dns.exe ..\dim\bin\dimtree.exe server.exe client.exe client.exe
set killlist=dimtree.exe server.exe client.exe
set dllslist=..\dim\bin\dim.dll

for %%i in (%filelist% %dllslist%) do if not exist %%i ( echo ERROR: %%i not found. & pause &
goto :EOF )

echo ************************************************************
echo ************************************************************
echo ************************************************************
echo *** DIM test: example of how to use DIM.
echo *** DIM is Distributed Information Manager.
echo *** See http://dim.web.cern.ch for details.
echo ************************************************************
echo ***
echo *** PLEASE PRESS ENTER TO STOP CLIENT-SERVER TESTING.
echo ***
echo ************************************************************
echo ************************************************************
echo ************************************************************

for %%i in (%killlist%) do start /wait /min taskkill /im %%i
for %%i in (%tasklist%) do ( start %%i & ping -n 2 127.0.0.1 1>nul 2>nul )
pause
for %%i in (%killlist%) do start /wait /min taskkill /im %%i
```

Пояснения:

1) Для доступности файлов **DIM** каталог **dim\bin** добавляется в **PATH**.
2) Переменная окружения **DIM_DNS_NODE** устанавливается в **localhost**.
   То есть для проверки используется локальный сервер имен **DNS**.
3) Команда **demo.bat** стартует программы:
   dns, dimtree, server, client
   и ждет нажатия клавиши.
4) При нажатии клавиши запущенные процессы завершаются.

Приложение 6. Скрипт DIMPORTS.CMD.

```
@echo off

rem ############################################################
rem Copyright(c) 2018 by Alexey Kuryakin, Sarov Russia 20181127.
rem This script uses to open TCP ports for DimServer connections
rem ############################################################
:Main
if "%~1" == "" ( call :Usage & goto :EOF )
:CheckUserAccess
net session 1>nul 2>nul && goto :AccessGranted
echo Access denied to user "%UserName%".
ping -n 2 127.0.0.1 1>nul 2>nul
goto :EOF
:AccessGranted
echo Access granted to user "%UserName%".
echo.
echo Enable firewall ports for DIM...
echo.
setlocal
call :CheckArguments %*
call :CheckInterface vboxnet0 vboxnet0found
if "%todo%" == "" ( call :Usage & goto :EOF )
if /I "%todo%" == "show" ( call :FirewallShowPortopening & goto :EOF )
for /L %%i in (2505,1,2505) do call :FirewallPortopening TCP %%i "DIM DNS"  %mode% "SUBNET" ALL
vboxnet0
for /L %%i in (5100,1,5108) do call :FirewallPortopening TCP %%i "DIM DATA" %mode% "SUBNET" ALL
vboxnet0
endlocal
goto :EOF

:FirewallPortopening
if "%~1" == "" goto :EOF & if "%~2" == "" goto :EOF & if "%~3" == "" goto :EOF & if "%~4" == "" goto
:EOF & if "%~5" == "" goto :EOF & if "%~6" == "" goto :EOF
if /I "%todo%" == "add"    netsh firewall add    portopening protocol = %~1 port = %~2 name = "%~1
%~2 - %~3" mode = %~4 scope = %~5 profile = %~6    1>nul && echo %todo% %~1 %~2 %~4 success, profile
%~6   || echo %todo% %~1 %~2 %~4 failed, profile %~6
if /I "%todo%" == "delete" netsh firewall delete portopening protocol = %~1 port = %~2
profile = %~6    1>nul && echo %todo% %~1 %~2     success, profile %~6   || echo %todo% %~1 %~2 %~4
failed, profile %~6
if "%~7" == "" goto :EOF & if "%vboxnet0found%" == "0" goto :EOF
if /I "%todo%" == "add"    netsh firewall add    portopening protocol = %~1 port = %~2 name = "%~1
%~2 - %~3" mode = %~4             interface = %~7  1>nul && echo %todo% %~1 %~2 %~4 success,
interface %~7 || echo %todo% %~1 %~2 %~4 failed, interface %~7
if /I "%todo%" == "delete" netsh firewall delete portopening protocol = %~1 port = %~2
interface = %~7  1>nul && echo %todo% %~1 %~2     success, interface %~7 || echo %todo% %~1 %~2 %~4
failed, interface %~7
goto :EOF

:CheckArguments
set todo=
set mode=ENABLE
:CheckArgumentsLoop
if "%~1" == "" goto :EOF
call :CheckArgumentsLoop %2 %3 %4 %5 %6 %7 %8 %9
if /I "%~1" == "ADD"      set todo=add
if /I "%~1" == "SHOW"     set todo=show
if /I "%~1" == "DELETE"   set todo=delete
if /I "%~1" == "ENABLE"   set mode=ENABLE
if /I "%~1" == "DISABLE"  set mode=DISABLE
goto :EOF

:FirewallShowPortopening
netsh firewall show portopening verbose = ENABLE
goto :EOF

:CheckInterface
if "%~1" == "" goto :EOF
if "%~2" == "" goto :EOF
set /a %~2=0
for /F %%j in ('ipconfig ^| find /i "%~1"') do set /a %~2+=1
goto :EOF

:Usage
echo.
echo Copyright^(c^) 2018 Alexey Kuryakin kouriakine@mail.ru
echo %~n0 - tool to add/delete/enable/disable DIM ports
```

```
echo DIM uses TCP ports 2505,5100..5108
echo see http://http://dim.web.cern.ch
echo.
echo Usage:
echo  %~n0               - help
echo  %~n0 show          - show all opened ports
echo  %~n0 delete        - delete opened   DIM ports
echo  %~n0 add           - add and enable  DIM ports
echo  %~n0 add enable    - add and enable  DIM ports
echo  %~n0 add disable   - add and disable DIM ports
echo.
goto :EOF

:SamplesToCopy
echo Delete ports from profile ALL...
netsh firewall delete portopening protocol = TCP port = 2505 profile = ALL
netsh firewall delete portopening protocol = TCP port = 5100 profile = ALL
netsh firewall delete portopening protocol = TCP port = 5101 profile = ALL
netsh firewall delete portopening protocol = TCP port = 5102 profile = ALL
echo Delete ports from interface vboxnet0...
netsh firewall delete portopening protocol = TCP port = 2505 interface = vboxnet0
netsh firewall delete portopening protocol = TCP port = 5100 interface = vboxnet0
netsh firewall delete portopening protocol = TCP port = 5101 interface = vboxnet0
netsh firewall delete portopening protocol = TCP port = 5102 interface = vboxnet0
echo Add ports to profile ALL...
netsh firewall add portopening protocol = TCP port = 2505 name = "TCP 2505 - DIM DNS" mode = ENABLE
scope = SUBNET profile = ALL
netsh firewall add portopening protocol = TCP port = 5100 name = "TCP 5100 - DIM DATA" mode = ENABLE
scope = SUBNET profile = ALL
netsh firewall add portopening protocol = TCP port = 5101 name = "TCP 5101 - DIM DATA" mode = ENABLE
scope = SUBNET profile = ALL
netsh firewall add portopening protocol = TCP port = 5102 name = "TCP 5102 - DIM DATA" mode = ENABLE
scope = SUBNET profile = ALL
echo Add ports to interface vboxnet0...
netsh firewall add portopening protocol = TCP port = 2505 name = "TCP 2505 - DIM DNS" mode = ENABLE
interface = vboxnet0
netsh firewall add portopening protocol = TCP port = 5100 name = "TCP 5100 - DIM DATA" mode = ENABLE
interface = vboxnet0
netsh firewall add portopening protocol = TCP port = 5101 name = "TCP 5101 - DIM DATA" mode = ENABLE
interface = vboxnet0
netsh firewall add portopening protocol = TCP port = 5102 name = "TCP 5102 - DIM DATA" mode = ENABLE
interface = vboxnet0
echo Firewall show firewall portopening...
netsh firewall show portopening verbose = ENABLE
goto :EOF
```

Примечание:

Команда **dimports.cmd** (**add**/**delete**/**show**) разрешает, запрещает или показывает правила сетевого экрана **Windows Firewall** для стандартных портов **DIM** (2505, 5100 … 5108), необходимых для запуска до трех **DIM** серверов. При необходимости номера и диапазон портов можно легко изменять.

Приложение 7. Шаблон спецификации DIM сервера.

Спецификация интерфейса DIM - сервера DIM_TASK/DEMO/GENDC/SERVER.

1. Общая информация о функции DIM сервера и его настройках

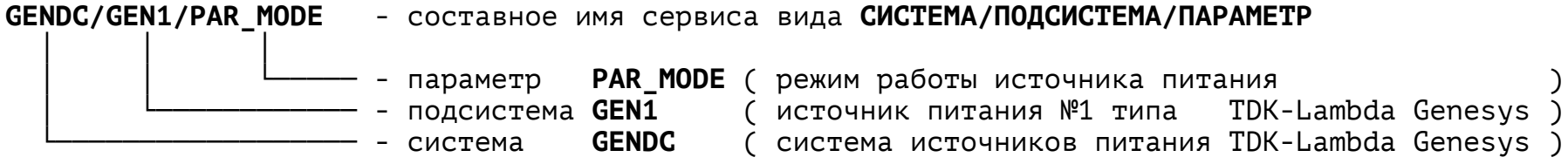
Основная функция сервера: обслуживание системы источников питания TDK-Lambda серии Genesys.

Параметр	Окружение	Значение	Комментарий
Имя сервера имен	**DIM_DNS_NODE**	По месту	Определяется сетевым администратором
Порт сервера имен	**DIM_DNS_PORT**	**2505**	Стандартное значение по умолчанию
Порты данных **DIM**	---	**5100,5101,5102,…,6000**	Стандартное значение по умолчанию
Имя **DIM** сервера	---	**DIM_TASK/DEMO/GENDC/SERVER**	**DIM_TASK** - общий префикс для имен **DIM** серверов **DEMO/GENDC** - условное имя системы/подсистемы **SERVER** - обозначение роли (**SERVER**/**CLIENT**)

2. Соглашения о наименовании DIM сервисов

Имена **DIM** сервисов имеют составной вид, содержащий идентификаторы (включающие знак подчеркивания «_», латинские буквы в верхнем регистре и цифры), разделенные символом слэш «**/**». Имена сервисов должны быть уникальными в пределах данной (локальной) сети. Для каждой системы/подсистемы следует использовать свой префикс, уникальный в пределах сети **DIM**. Число компонентов имени может быть разным, например, **СИСТЕМА/ПОДСИСТЕМА/СТОЙКА/УСТРОЙСТВО/ПАРАМЕТР**. При помещении в сервис нескольких параметров допустимо использовать имена вида **ПАРАМЕТР1+ПАРАМЕТР2**, например, **RAMP_A+V**.

Например:



3. Таблица публикуемых DIM сервисов сервера DIM_TASK/DEMO/GENDC/SERVER

№	Имя сервиса	Тип сервиса	Комментарий
1	GENDC/GEN1/BT_AST	L	Кнопка режима автостарта. Бит 0 – состояние кнопки AST (autostart). Бит 1 – состояние параметра AST 0/1=OFF/ON.
2	GENDC/GEN1/BT_FLD	L	Кнопка режима защиты по току. Бит 0 – состояние кнопки FLD (foldback protection). Бит 1 – состояние параметра FLD 0/1=OFF/ON.
3	GENDC/GEN1/BT_OUT	L	Кнопка подключения выхода (нагрузки). Бит 0 – состояние кнопки OUT (output). Бит 1 – состояние параметра OUT 0/1=OFF/ON.
4	GENDC/GEN1/ERROR_CNT	D	Счетчик ошибок. Содержит число обнаруженных ошибок связи, протокола обмена, задания параметров и т.д.
5	GENDC/GEN1/ID_DATE	C	Строка даты последней проверки источника питания. Например: 2019/06/25
6	GENDC/GEN1/ID_IDN	C	Строка идентификатора модели источника питания. Например: LAMBDA,GEN6-100
7	GENDC/GEN1/ID_REV	C	Строка версии прошивки источника питания. Например: 1U1K:5.1.3
8	GENDC/GEN1/ID_SN	C	Строка серийного номера источника питания. Например: 011B431-0003
9	GENDC/GEN1/PAR_FENA	L	Регистр разрешения для ошибок (fault enable register). Управляет разрешением генерации событий (SRQ) при ошибках. Подробнее см. документацию TDK-Lambda.
10	GENDC/GEN1/PAR_FEVE	L	Регистр событий по ошибкам (fault event register). Содержит информацию о событиях, связанных с ошибками. Подробнее см. документацию TDK-Lambda.
11	GENDC/GEN1/PAR_FR	L	Регистр ошибок (fault register). Содержит текущий статус ошибок разных типов.

			Подробнее см. документацию TDK-Lambda.
12	GENDC/GEN1/PAR_MC	D	Измерение тока, Ампер (measure current).
13	GENDC/GEN1/PAR_MODE	L	Режим работы источника питания: 0/1/2/3 = OFF/CV/CC/ERROR. CV (constant voltage) - режим стабилизации напряжения. CC (constant current) - режим стабилизации тока.
14	GENDC/GEN1/PAR_MV	D	Измерение напряжения, Вольт (measure voltage).
15	GENDC/GEN1/PAR_MW	D	Измерение мощности, Ватт (measure watt).
16	GENDC/GEN1/PAR_OVP	D	Уровень защиты от перенапряжения (over voltage protection), Вольт.
17	GENDC/GEN1/PAR_PC	D	Установка тока стабилизации (programmed current), Ампер.
18	GENDC/GEN1/PAR_PV	D	Установка напряжения стабилизации (programmed voltage), Вольт.
19	GENDC/GEN1/PAR_RMT	L	Режим удаленного доступа: 0/1/2=LOC/REM/LLO. Подробнее см. документацию TDK-Lambda.
20	GENDC/GEN1/PAR_SR	L	Регистр статуса (status register). Подробнее см. документацию TDK-Lambda.
21	GENDC/GEN1/PAR_UVL	D	Уровень защиты по низкому напряжению (under voltage limit), Вольт.
22	GENDC/GEN1/POLL_ENABLE	L	Разрешение опроса COM порта, 0/1=OFF/ON.
23	GENDC/GEN1/POLL_RATE	D	Частота опроса COM порта (опросов в секунду), Герц.
24	GENDC/GEN1/RAMP_A+V	L:2	1)Состояние перехода по току (ramping ampers), 0/1=OFF/ON. 2)Состояние перехода по напряжению (ramping volts), 0/1=OFF/ON.
25	GENDC/GEN1/RATE_A+V	D:2	1)Расчетная скорость перехода по току, Ампер/сек. 2)Расчетная скорость перехода по напряжению, Вольт/сек.

Примечания:

1. Допустимые типы сервиса: **L**/**I**/**D**/**C**=Long/Integer/Double/Char. Типы Long и Integer - синонимы.
2. При наличии в сервисе нескольких параметров используется формат вида **L:2;D:3** (2×Long и 3×Double).
3. В таблице указан список параметров для одной подсистемы GENDC/GEN1. Для других подсистем того же типа (например, GENDC/GEN2, с заменой GEN1 на GEN2) список сервисов аналогичен.