

Программный комплекс для создания диспетчерских
информационно-управляющих систем реального
времени
«КОТМИ-2010»

Руководство программиста

версия 1.7
Москва 2009г.



СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	2
1. АРХИТЕКТУРА СИСТЕМЫ С ТОЧКИ ЗРЕНИЯ ПРОГРАММИСТА	3
2. НАПИСАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ. ПРОТОКОЛ ПРИКЛАДНОГО УРОВНЯ MDE.....	5
2.1. ОБЗОР.....	5
2.2. ИСПОЛЬЗОВАНИЕ DLL-ИНТЕРФЕЙСА (Oicmde.dll).....	5
2.2.1. Обзор.....	5
2.2.2. Подключение к серверу	6
2.2.3. Элементарные типы данных	8
2.2.4. Управление данными	9
2.2.4.1. Таблицы в памяти	9
2.2.4.2. Создание объекта.....	10
2.2.4.3. SQL – запросы.....	12
2.2.4.4. Встроенные процедуры.....	15
2.2.4.5. Изменение записей НСИ.....	17
2.2.5. Управление событиями.....	17
2.3. ИСПОЛЬЗОВАНИЕ OLE-ИНТЕРФЕЙСА (Scdsys.ocx).....	18
2.3.1. Обзор.....	18
2.3.2. Канал связи с сервером (ScadaCli)	19
2.3.2.1. Интерфейс IScadaForm.....	19
2.3.2.2. Интерфейс IScadaCli	20
2.3.2.3. События объекта.....	23
2.3.2.4. Установка и разрыв соединения.....	23
2.3.3. Абонент (ScadaAbo)	24
2.3.3.1. Интерфейс IDataRec	24
2.3.3.2. Интерфейс IScadaAbo.....	24
2.3.3.3. События объекта.....	27
2.3.4. Дополнительные возможности.....	28
2.3.4.1. Таблицы в памяти.IMDAArray, IMDARec	28
2.3.4.2. Хранилище данных. IDataBag	31
3. АРМ ПОЛЬЗОВАТЕЛЯ.....	33
3.1. НАРАЩИВАНИЕ ФУНКЦИОНАЛЬНОСТИ	33
3.1.1. Обзор.....	33
3.1.2. Интерфейс IScdDisp	33
3.1.3. Интерфейс IScadaObj	33
3.1.4. События объекта.....	35
3.1.5. Меню. IScadaMenu	35

ВВЕДЕНИЕ

Что делать, если стандартные средства настройки и администрирования системы недостаточны для решения специфических информационных запросов организации?

Как расширить функциональность АРМ клиента, разработать и подключить к нему новые модули с дополнительными возможностями?

Ответы на эти и многие другие вопросы даются в данном руководстве.

Более того, при разработке самой системы «КОТМИ-2010» использовались именно эти, изложенные ниже, интерфейсы и методики.

1. АРХИТЕКТУРА СИСТЕМЫ С ТОЧКИ ЗРЕНИЯ ПРОГРАММИСТА

Система «КОТМИ-2010», может быть представлена как совокупность нескольких, функционально выделенных подсистем. Взаимодействие подсистем между собой осуществляется с помощью оригинального протокола прикладного уровня (*Message Data Exchange, MDE*) разработанного поверх TCP/IP.

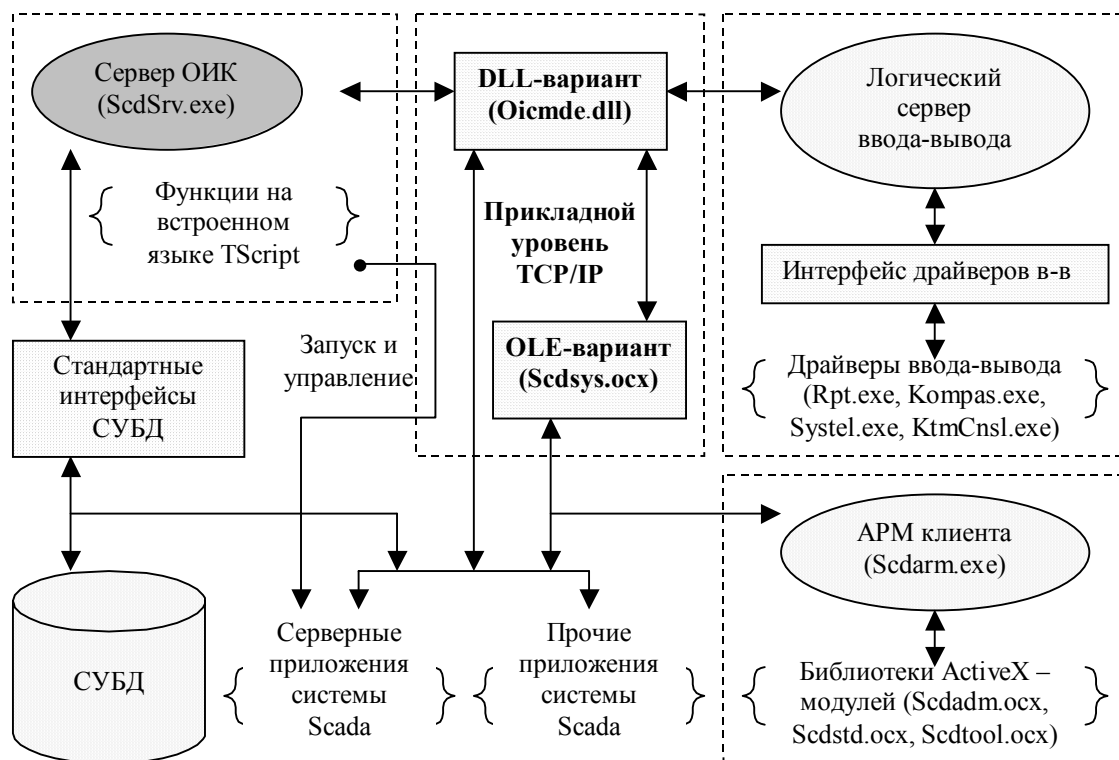


Рисунок 1 Архитектура системы «КОТМИ-2010»

Сервер ОИК - сердце системы.

Архитектурно его можно рассматривать как “сервер приложений” в трехуровневой модели “клиент-сервер”. С помощью протокола MDE сервер обеспечивает:

- 1) работу клиентских приложений с реляционной БД системы;
- 2) формирование архивов БД реального времени и доступ к этим архивам;
- 3) обработку, генерацию и оповещение клиентских приложений о событиях возникающих в системе;
- 4) параллельное функционирование подсистемы расчетов и управления, с использованием процедур, написанных на встроенном языке *TScript*;
- 5) запуск и управление “серверными” программами системы.

В качестве реляционной БД может быть использована любая БД, доступ к которой может быть осуществлен с помощью ADO или ODBC. Если SQL сервер имеет свой API - то в целях эффективности рекомендуется использование возможностей этого API напрямую (Interbase, MS SQL, Oracle). В базовом варианте поставки используется интерфейс ADO и БД структуры MS Access.

Кроме сервера ОИК (и драйверов устройств ввода-вывода), все другие программы системы «КОТМИ-2010», в терминологии трехуровневой модели “клиент-сервер”,

являются типичными “клиентами”. Территориально “клиенты” могут располагаться на одной или нескольких ЭВМ локальной сети, в том числе на одной машине с сервером ОИК.

Клиентские программы, зарегистрированные в таблице НСИ “Т_SPRG”, условно можно назвать “серверными”. Такие программы могут запускаться и контролироваться непосредственно самим сервером ОИК.

Взаимосвязь программ «клиентов» и сервера ОИК осуществляется с помощью протокола прикладного уровня TCP/IP - MDE. Именно клиентские приложения являются основными поставщиками и потребителями информации в системе.

АРМ пользователя - клиентское MDI-приложение, реализующее функции ActiveX-контейнера ОС Windows. Все функциональные модули АРМ представляют собой ActiveX-объекты, размещенные в библиотеках. Библиотеки, входящие в стандартную поставку обеспечивают:

- администрирование БД НСИ и, следовательно, управление функционированием системы;
- формирование и просмотр документов, схем, ретроспективы;
- оповещение о событиях и их квитирование в рамках заданной зоны ответственности;
- возможности конфигурирования и индивидуальной настройки видов АРМов;
- ведение специального календаря, отображение паспортов и многое другое.

Расширение функциональности АРМа возможно за счет модификации существующих или разработки и подключения новых библиотек с дополнительными возможностями.

Все ActiveX-модули этих библиотек должны поддерживать интерфейс **IscadaObj** или реализовать соответствующие свойства и методы в своем Dispatch-интерфейсе.

Итак, подытожим выше сказанное. Как программист может расширить возможности системы. На рисунке 1 в фигурных скобках заключены части системы, которые могут наращиваться путем программирования. Это:

- 1) написание процедур обработки на языке **TScript**;
- 2) подключение устройства ввода-вывода с помощью нового драйвера для сервера ввода-вывода системы (C, Pascal и т.п.);
- 3) расширение функциональных возможностей АРМ пользователя новыми библиотеками ActiveX-модулей (C, Pascal, VB и т.п.);
- 4) написание самостоятельного полноценного клиентского приложения, использующего протокол MDE для связи с сервером ОИК. Наравне с программами, написанными на традиционных языках программирования, такие клиенты могут быть разработаны с использованием приложений MS Office, MS Explorer и т.п.

Есть еще один способ создания клиента системы «КОТМИ-2010». Если организация использует в качестве СУБД SQL-сервер (Interbase, Oracle, MS SQL и т.п.) то, при соответствующем расширении структуры БД, возможно создание полноценного клиента «КОТМИ-2010», использующего исключительно стандартные интерфейсы БД. Данные возможности относятся к так называемым «корпоративным» расширениям системы и в стандартную поставку не входят.

2. НАПИСАНИЕ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ. ПРОТОКОЛ ПРИКЛАДНОГО УРОВНЯ MDE

2.1. ОБЗОР

Протокол прикладного уровня MDE (Message Data Exchange) обеспечивает соединение клиента и сервера ОИК поверх сетевого протокола TCP/IP. Со стороны клиента предоставляются следующие возможности:

- 1) подключение к серверу (создание сессии);
- 2) управление данными;
- 3) управление событиями.

Для подключения к серверу ОИК требуется указать сетевое имя компьютера, на котором сервер расположен, имя и пароль пользователя, зарегистрированные в системе «КОТМИ-2010».

Управление данными включает:

- получение результатов SQL-запросов к данным НСИ;
- модификацию записей таблиц НСИ;
- чтение-запись архивных данных;
- выполнение специальных процедур, встроенных в сервер ОИК.

Управление событиями: - оповещение клиента о возникновении интересующего его события. Клиент может сформировать для себя требуемое подмножество событий, указав соответствующие энергообъекты и коды событий:

2.2. ИСПОЛЬЗОВАНИЕ DLL-ИНТЕРФЕЙСА (Oicmde.dll)

2.2.1. Обзор

Весь базовый механизм протокола MDE реализован в файле OicMde.dll. Для работы с сервером ОИК, требуется установить соединение и создать объекты для работы с данными и событиями.

Однотипных объектов может быть несколько. Каждый такой объект обеспечивает в рамках одной сессии полную функциональность и независимость работы от других объектов того же типа. В большинстве случаев для нормальной работы клиента достаточно иметь по одному объекту данных и событий.

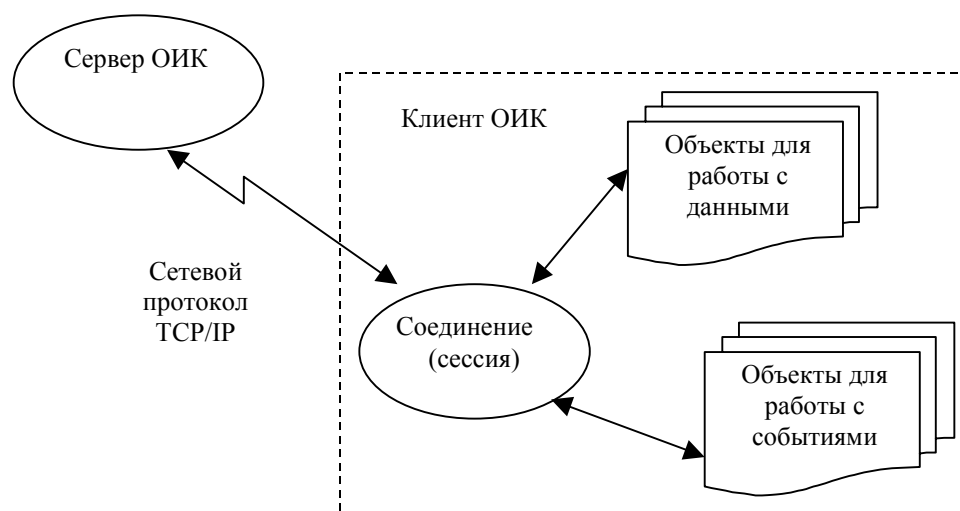


Рисунок 2 Архитектура клиента системы «КОТМИ-2010»

Все типы объектов с точки зрения программиста устроены по одному принципу:

- методы объекта определяют его функциональность;
- обработчики событий позволяют реагировать на изменение состояния объекта или контролировать активный процесс обработки данных.

Процедуры обработки событий связываются с объектом при его создании.

Основные интерфейсы для работы с библиотекой oicmde.dll описаны в файлах:

- Mdsoc.h – механизм взаимодействия по TCP/IP;
- Mdrec.h – записи с полями. Элементарная единица данных;
- MdRecArr.h – таблицы в памяти. Массивы упорядоченных записей;
- Mde.h – базовый механизм формирования протоколов прикладного уровня;
- Mdeses.h – соединение с сервером. Создание сессии;
- Mdedb.h – протокол работы с данными;
- Mdevn.h – протокол для работы с сообщениями.

Интерфейсы доступа из Delphi описаны в файле – Mde.pas.

Ниже последовательно описаны все этапы работы с сервером ОИК, необходимые для написания клиентского приложения с использованием Oicmde.dll. Примеры приводятся на языке C. Сам проект находится в директории Scd\Client\Examples\Vc\WriteHVed и реализует алгоритм записи архивов часовой ведомости. Файлы mc_ses.cpp, mc_db.cpp и mc_evn.cpp могут служить шаблонами для написания своего клиента ОИК.

2.2.2. Подключение к серверу

Первоначальный этап работы с библиотекой Oicmde.dll – инициализация внутренних структур библиотеки. В качестве параметра функции требуется задать максимальное количество одновременно открытых сессий данного приложения с серверами ОИК.

```
#include "mde.h"
MDError Err;
Err = MDEInit(4);
If (Err) return; // Ошибка, если Err != 0;
```

Следующий этап – подключение к серверу ОИК. Сразу после открытия сессии, имеет смысл создать программные объекты (не путать с OLE-объектами) для работы с данными и сообщениями.

```
#include "mc_ses.h"
#include "mc_db.h"
#include "mc_evn.h"
MDESes SesId;
MDEObj ObjDB, ObjEvn;
// Создание сессии и объектов
// для работы с данными и сообщениями
SesId = MD_SesOpen("Oic1", "Mid", "1");
if(!SesId) return FALSE;
ObjDB = MDB_CliObjNew(SesId);
if(!ObjDB) return FALSE;
ObjEvn = MDV_CliObjNew(SesId);
if(!ObjEvn) return FALSE;
```

После выполнения требуемых действий заключительный этап – разрыв связи с сервером. Перед разрывом соединения рекомендуется удалить все ранее созданные объекты.

```

#include "mc_ses.h"
#include "mc_db.h"
#include "mc_evn.h"
MDESes CliSesId;
MDEObj CliObjDB, CliObjDBEvn;
// Удаление объектов и завершение сессии
MDV_CliObjDel(ObjEvn); ObjEvn = 0;
MDB_CliObjDel(ObjDB); ObjDB = 0;
MDSesClose(SesId); SesId = 0;

```

Любой объект в MDE представляет собой сущность, характеризуемую своим идентификатором. Используя идентификатор в качестве параметра, можно вызывать функции-методы, свойственные данному типу объекта. Объект же, в свою очередь, информирует свое окружение о внутренних событиях с помощью вызова функций-обработчиков, которые должны быть связаны с ним в момент создания.

Рассмотрим, в качестве примера, реализацию функции MD_SesOpen, осуществляющую непосредственное подключение клиента к серверу ОИК (см. mc_ses.cpp).

```

#include "mc_ses.h"
#include "mc_db.h"
#include "mc_evn.h"
// Предварительное описание функций обработчиков событий сессии
static void onSesClose(MDESes SesId);
static void onSesAbort(MDESes SesId);
static void onSesUser(MDESes SesId);
// Структура с адресами обработчиков событий сессии
SesCliEvent SesEvent =
{
    &onSesClose,
    &onSesAbort,
    &onSesUser
};
// Подключение клиента к серверу
MDESes MD_SesOpen(char *SrvAddr, char *UserName, char *UserPsw)
{
    MDESes SesId = 0;
    bool NoUser;
    // Причина неудачного подключения может быть уточнена
    // с помощью NoUser (TRUE – неправильное имя или пароль,
    // FALSE – сервер не найден;
    if (!MDSesOpen(SrvAddr, // Сетевое имя ЭВМ или IP-адрес
        UserName, // Имя и пароль пользователя
        UserPsw, // (с учетом регистра)
        &SesEvent, // Адреса функций-обработчиков
        &SesId, // (out) Идентификатор сессии
        &NoUser)) { return 0; } // (out)
    return SesId;
}
// Обработка сообщений SES, реализация
// Закрывание сессии (сессия явно закрыта сервером)
void onSesClose(MDESes SesId)
{
}
// Аварийное закрытие сессии (разрыв виртуального канала)
void onSesAbort(MDESes SesId)

```



```

{
    onSesClose (SesId) ;
}
// Изменение характеристик пользователя сессии на сервере
void onSesUser (MDESes SesId)
{
}

```

Открытие сессии в функции MD_SesOpen осуществляется с помощью обращения к функции MDsesOpen, описанной в mdeses.h. Одним из параметров ей передается структура с адресами функций-обработчиков событий. В данном случае возможных событий три: закрытие сессии сервером, аварийный разрыв каната и изменение характеристик пользователя в системе (например, пароля или имени).

Аналогичным образом осуществляется взаимодействие с объектами двух других типов: данных и событий. Методы и обработчики для них описаны ниже в следующих разделах.

2.2.3. Элементарные типы данных

В пользовательских функциях и структурах MDE используются следующие базовые типы данных (см. mdrec.h):

№	Тип	Описание
1	mdr_Tbool	Логический (True, False)
2	mdr_Tbyte	Байт, без знаковое целое (0..255)
3	mdr_Tsmall	Целое, 2 байта (-32,768..32,767)
4	mdr_Tint	Целое, 4 байта (-2,147,483,648..2,147,483,647)
5	mdr_Tfloat	Вещественное, 4 байта (3.4E +/- 38 (7 digits))
6	mdr_Tdouble	Вещественное, 8 байт (1.7E +/- 308 (15 digits))
7	mdr_Tchar	Текст фиксированной длины (1..255 символов)
8	mdr_Tmemo	Текст произвольной длины
9	mdr_TunixDT	Время в UNIX формате, Sec, 4 байта (mm-dd-yyuu hh:mm:ss)
10	mdr_TUnixMDT	Время в UNIX формате, Sec и Msec, 8 байт (mm-dd-yyuu hh:mm:ss.ms)
11	mdr_Tcurrency	Деньги, 8 байт (-922337203685477.5808 .. 922337203685477.5807)

В mdrec.h файле основные типы описаны как:

```

typedef bool          MDRBool;
typedef char          MDRByte;
typedef short         MDRSmall;
typedef long          MDRInt;
typedef float         MDRFloat;
typedef double        MDRDouble;
typedef char*         MDRBlob;
typedef char*         MDRMemo;
typedef char*         MDRChar;
typedef CURRENCY MDRCurrency; // LONGLONG
typedef long          MDRUnixDT;
typedef double        MDRUnixMDT;

```

Соответственно универсальный тип данных и описание поля представлены как:

```
union MDRVar
{
    MDRBool          vBool;
    MDRByte          vByte;
    MDRSmall         vSmall;
    MDRInt           vInt;
    MDRFloat         vFloat;
    MDRDouble        vDouble;
    MDRText          vText;
    MDRUnixDT        vUnixDT;
    MDRUnixMDT       vUnixMDT;
    MDRCurrency      vCurrency;
};

struct MDRFieldDef
{
    char FldName[MDR_MAX_NAME + 1]; /* Наименование поля */
    int FldSize;                    /*
Размер поля */
    MDRFldType FldType;             /* Тип поля */
};
```

И, наконец, описание структуры, которая используется при доступе к полям записи в обработчиках событий MDE, выглядит следующим образом:

```
/* Вариантное поле */
struct MDRFieldVal
{
    MDRFieldDef Fld;
    MDRVar Val;
    bool IsNull;
};
```

Данные, описываемые с помощью этой структуры, предназначены только для чтения и изменению не подлежат.

2.2.4. Управление данными

2.2.4.1. Таблицы в памяти

Таблицы в памяти представляют собой упорядоченный массив записей одинаковой структуры (см. mddarray.h). Понятие структуры и функции доступа к полям записи описаны в файле MdRec.h.

Типы полей соответствуют основным типам данных MDE.

В одной записи может быть до 255 полей. Доступ к полям записи осуществляется в два этапа:

- 1) получение описателя поля с помощью функций MDRTypFld или MDRTypFldN;
- 2) чтение или изменение значения поля функциями MDRGet*, MDRSet*;

В файле MdRecArr.h описаны функции и структуры для работы с таблицей в памяти.

Таблицу можно создать и удалить функциями MDACreate и MDADestroy. При этом само описание таблицы будет размещено в структуре MDArray, передаваемой в качестве параметра.

Формирование структуры полей производится с помощью MDALdAdd. Данная функция может быть вызвана несколько раз подряд. Новые поля будут добавлены к списку уже существующих.

Ключи (Key) или индексы – это списки, различным образом упорядочивающие записи таблицы. Для одной таблицы можно задать до 16 ключей и каждый ключ может содержать до 6 полей таблицы включительно. Добавление или удаление ключей осуществляется функциями MDALKeyAdd и MDALKeyDel динамически. Т.е. добавление ключа формирует новый упорядоченный список на записях таблицы. Удаление – уничтожает список, но не записи таблицы. При добавлении записей в таблицу для которой ключи не определены, автоматически создается ключ с пустым списком полей.

Функции MDALRecNew, MDALRecEdit создают или копируют запись для редактирования. После изменения полей MDALRecPost сохранит изменения в таблице и перестроит индексы (если они не были созданы с признаком idxNonMaintained). Функция MDALRecDel удаляет запись из таблицы, или отменяет изменения, если она вызывается для дескрипторов, полученных с помощью MDALRecNew, MDALRecEdit.

Порядковый доступ к записи в ключевом списке возможен с помощью функции MDALRec, а. поиск по ключу – MDALRecSeek.

2.2.4.2. Создание объекта

Интерфейс объекта данных обеспечивает выполнение трех видов запросов к серверу:

- 1) SQL – запрос;
- 2) запрос на запуск серверной процедуры;
- 3) команды на изменение данных в таблицах НСИ.

Результатом выполнения запроса первого или второго типов, может быть таблица с произвольным количеством строк. Результат может быть получен как синхронно, так и асинхронно по отношению к запросу. Команды третьего вида изменяют данные в таблицах НСИ и результатов не возвращают. Все запросы завершаются функцией «mMDBCliPost», фиксирующей конец «тела» текущей команды.

Запросы формируются и выполняются в составе блока (пакета). Один блок может содержать произвольное количество разнородных запросов и выполняется как одна транзакция. В случае возникновения ошибочной ситуации в процессе выполнения блока команд на сервере, клиент информируется об ошибке, а изменения НСИ, сделанные в результате исполнения предыдущих запросов блока, “откатываются” на момент начала транзакции.

Все этапы процесса исполнения блока команд на сервере, отражаются на клиенте через вызов событийных процедур, связанных с объектом в момент создания.

Ниже, в качестве примера, рассматривается реализация функций MDB_CliObjNew и MDB_CliObjDel, которые соответственно создают и освобождают объект для работы с БД сервера ОИК (см. mc_db.cpp). С каждым таким объектом может быть связано произвольное long-значение, которое в дальнейшем может использоваться по усмотрению разработчика.

```
#include "mc_db.h"
// Предварительное описание функций обработчиков событий
static void onObjAbort(MDESes SesId, MDEObj ObjId);
static void onBlockBeg(MDEObj ObjId, long ObjData, long UserIdn);
static void onBlockEnd(MDEObj ObjId, long ObjData, bool IsCommit);
static void onError(MDEObj ObjId, long ObjData, int Err, char* Prm);
static void onProcBeg(MDEObj ObjId, long ObjData, char *Proc,
    char *Prm, MDArray **Tbl, bool *ReplaceEq);
```

```

static void onProcEnd(MDEObj ObjId, long ObjData, MDArray *Tbl);
static void onSqlBeg(MDEObj ObjId, long ObjData,
    char *Sql, char *Prm, MDArray **Tbl, bool *ReplaceEq);
static void onSqlEnd(MDEObj ObjId, long ObjData, MDArray *Tbl);
static void onValRow(MDEObj ObjId, long ObjData, int FldCount,
    MDRFieldVal *FldArray);
typedef struct
{
    // Структура данных разработчика
    // . . .
} DataDscCli;
// Структура с адресами обработчиков событий сессии
static MDBCliEvent ObjEvent =
{
    &onBlockBeg,      // Начало обработки блока (пакета) на сервере
    &onBlockEnd,      // Завершение блока (успешное или аварийное)
    &onValRow,        // Очередная строка результата SQL- или PROC-запроса
    &onError,         // Ошибка, выполнение пакета, прекращается
    &onProcBeg,       // Серверная процедура. Начало записей результата
    &onProcEnd,       // Серверная процедура. Конец результата
    &onSqlBeg,        // SQL - запрос. Начало записей результата
    &onSqlEnd,        // SQL - запрос. Конец результата
};
// Создание объекта для работы с данными
MDEObj MDB_CliObjNew(MDESes SesId)
{
    MDEObj ObjId;
    DataDscCli *MyData;
    MyData = (DataDscCli*)malloc(sizeof(DataDscCli));
    if (!MDBCliObjNew(SesId,      // Идентификатор объекта сессии
        &onObjAbort,           // Обработчик аварийного закрытия
        &ObjEvent,             // Обработчики событий обработки
        long)MyData,           // С объектом можно связать и
        &ObjId))               // в дальнейшем любое long-значение
    {
        free(MyData); return 0;
    }
    // Инициализация MyData
    // . . .
    return ObjId;
}

void MDB_CliObjDel(MDEObj ObjId)
{
    DataDscCli *MyData = NULL;
    MDBCliObjDel(ObjId,
        (long*)&MyData); // (out) long-значение, заданное при создании
    if(MyData != 0)
    {
        free(MyData);
    }
}

```

2.2.4.3. SQL – запросы

С помощью механизма SQL-запросов осуществляется выборка требуемой информации из таблиц НСИ на сервере. Расположенный ниже фрагмент программы, извлекает из таблицы телеизмерений (T_TI) идентификаторы ТИ. Запрос осуществляется синхронно, т.е. после завершения функции «mMDBCliBlockEnd» данные клиентом будут уже получены. Если бы третий параметр «mMDBCliBlockEnd» был установлен в FALSE, то выполнение программы продолжалось без ожидания результата (асинхронно).

```

// Чтение идентификаторов ТИ.
BOOL GetIdTi(MDEObj CliObjDB)
{
    mMDBCliBlockBeg(          // Начало блока запроса:
        CliObjDB,            // идентификатор объекта данных
        0);                  // произвольное long-значение
    mMDBCliSql(CliObjDB,      // Начало SQL-запроса:
        "SELECT TI_ID FROM T_TI", // строка SQL-запроса
        "NAME=T_TI");        // имя таблицы
    mMDBCliPost(CliObjDB);    // Конец тела SQL-запроса
    mMDBCliBlockEnd(CliObjDB, // Конец блока запроса:
        TRUE, // Подтверждение (или откат) транзакции
        TRUE); // Ожидание завершения блока или асинхронное продолжение
    return TRUE;
}

```

Сам процесс обработки событий клиентом при получении данных будет происходить в следующей последовательности:

OnBlockBeg – начало блока результата;

OnSqlBeg (onProcBeg) – начало данных SQL (PROC) – запроса

onValRow (*)– записи результирующей таблицы SQL (PROC) - запроса

OnSqlEnd (onProcEnd) – завершение данных SQL (PROC) - запроса

OnBlockEnd – завершение блока результата.

Как уже отмечалось выше, в одном блоке запроса могут содержаться результаты нескольких SQL- или PROC-запросов к серверу.

Здесь следует наверно добавить, что при вызове «mMDBCliSql» можно дополнительно задать еще три параметра:

- 1) Tbl – виртуальную таблицу, куда будет размещен результат запроса;
- 2) ReplaceEq – режим обработки записей совпадающих по уникальному ключу
- 3) DoEvent – требование активации процедуры-обработчика событий (onSqlBeg, onValRow, onSqlEnd) даже при наличии Tbl.

Если в подставляемой виртуальной таблице отсутствует структура полей, то она формируется автоматически в соответствии со структурой полей записи результата, иначе в таблицу заносятся только те поля, имена которых совпадают.

```

void onBlockBeg(MDEObj ObjId, // Идентификатор объекта данных
    long ObjData, // данные, связанные с объектом при создании
    long UserIdn) // long-значение заданное в mMDBCliBlockBeg
{
}
void onBlockEnd(MDEObj ObjId, long ObjData,
    bool IsCommit)// результат завершения блока запроса
{
}

```

Обработчики событий onSqlBeg и onSqlEnd (onProcBeg, onProcEnd) вызываются, если в функции «mMDBCliSql» (mMDBCliProc) не заданна виртуальная таблица или параметр DoEvent = TRUE. В «onSqlBeg» (onProcBeg) можно изменить указатель виртуальной таблицы и режима обработки совпадающих записей. Обработчик события «onValRow» вызывается только при отсутствии виртуальной таблицы.

В обработчике события onSqlBeg (onProcBeg) можно заранее определить структуру полей записей результата запроса. Для этого используются функции:

```

MDBCDef mMDBCliFldNum(MDEObj ObjId, unsigned *Num);
MDBCDef mMDBCliFld(MDEObj ObjId, int Fld, MDRFldType *Type,
unsigned *Size, char *Name);

```

Первая сообщает количество полей в записи, вторая – характеристики указанного поля. Индексы полей начинаются с 0.

```

/* Начало и конец списка строк результата SQL-запроса */
void onSqlBeg(MDEObj ObjId, long ObjData,
char *Sql,          // текст SQL-запроса
char *Prm,          // текст параметров
MDArray **Tbl,      // таблица в памяти (можно изменить)
bool *ReplaceEq)    // замена совпадающих строк в таблице
{
    DataDscCli* MyData;
    MyData = (DataDscCli*)ObjData;
    // Определение имени запроса
    if (!(NameReq(ObjId, ObjData, Prm))) return;
}

void onSqlEnd(MDEObj ObjId, long ObjData, MDArray *Tbl)
{
    DataDscCli *MyData;
    MyData = (DataDscCli*)ObjData;
}

```

Обработчик «onValRow» вызывается для каждой записи, полученной как результат SQL- или PROC-запроса (если не определена виртуальная таблица). Каждая запись представляет собой массив полей указанной длины. В общем случае, разные записи одного запроса могут содержать разное количество полей, поля могут отличаться типами. Однако все SQL- и большинство PROC-запросов возвращают записи одинаковой структуры.

```

/* Строка результатов запроса */
void onValRow(MDEObj ObjId, long ObjData,
int FldCount,          // Количество полей в записи
MDRFieldVal *FldArray) // Массив полей записи
{
    DataDscCli* MyData;
    int i;
    VntiDsc* Vnti;
    MyData = (DataDscCli*)ObjData;
    if (strcmp(*(MyData -> Name), "OBJ") == 0)
    { MyData->Obj = (ObjDsc*)calloc(1, sizeof (ObjDsc)); }
    if (strcmp (*(MyData -> Name), "VNTI_H") == 0)
    {
        MyData -> Vnti = (VntiDsc*)calloc(1, sizeof (VntiDsc));
        InitializeCriticalSection (&(MyData->Vnti->g_CritSecVntiDsc));
    }
    if (strcmp (*(MyData -> Name), "CODEVENT") == 0)
    { MyData -> EvCod = (EvCodDsc*)calloc(1, sizeof (EvCodDsc)); }
    if ((strcmp (*(MyData -> Name), "VAL_TI") == 0) ||
        (strcmp (*(MyData -> Name), "VAL_PTI") == 0) ||
        (strcmp (*(MyData -> Name), "VAL_VNTI") == 0))
    { MyData->Val = (ValDsc*)calloc(1, sizeof (ValDsc)); }
    //////////////////////////////////////
    for (i=0; i<FldCount; ++i)
        WriteValField(ObjData, (FldArray + i));
}

```

```

////////////////////////////////////
if (strcmp (*(MyData -> Name), "OBJ") == 0)
{
    MyData->ObjList->SetAt((WORD)MyData->Obj->m_lObjId, MyData-
    >Obj);
    return;
}
if (strcmp (*(MyData -> Name), "VNTI_H") == 0)
{
    MyData->VntiList->SetAt((DWORD)MyData->Vnti->m_lId,
    MyData->Vnti);
    return;
}
if (strcmp (*(MyData -> Name), "CODEVENT") == 0)
{
    MyData->EventCodList->SetAt((WORD)MyData->EvCod->m_lEvCodId,
    MyData->EvCod);
    return;
}
if ((strcmp (*(MyData -> Name), "VAL_TI") == 0))
{ MyData->TiVal->SetAt((DWORD)MyData->Val->m_nId, MyData->Val); }
if ((strcmp (*(MyData -> Name), "VAL_PTI") == 0))
{ MyData->PtiVal->SetAt((DWORD)MyData->Val->m_nId, MyData->Val); }
}
if ((strcmp (*(MyData -> Name), "VAL_VNTI") == 0))
{
    Vnti = GetPtrVntiCurr ();
    MyData -> VntiVal -> SetAt( (DWORD)Vnti -> m_lId, MyData ->
    Val);
}
}

```

2.2.4.4. Встроенные процедуры

Принципы работы с PROC-запросами (встроенными процедурами) в основном аналогичны принципам описанным выше для SQL-запросов. В некотором смысле SQL-запрос является частным случаем реализации PROC-запроса.

В отличие от SQL-запросов PROC-запросы позволяют изменять данные на сервере и получать требуемую информацию, как результат специальной (не обязательно SQL) обработки

На сервере реализован ряд вполне конкретных процедур, позволяющих работать с архивами БД РВ, синхронизировать время, управлять распределением нагрузки между резервными машинами и т.п. Полный список серверных процедур, их назначение и описание параметров приведен в приложении А.

Ниже представлен пример чтения значений из архива ТИ. Имя серверной процедуры отвечающей за чтение архивных данных – «READ_ARCH». Имя интересующего архива указывается в параметре «TABLE_NAME». В список параметров можно добавить ряд дополнительных, которые не будут учитываться на сервере, но могут помочь на клиенте при обработке результатов запроса. Как отмечалось выше, строки наименования и параметров без изменений будут переданы обработчику события onProcBegin (onSqlBegin). Отдельные параметры в строке должны быть разделены между собой символами «\r\n». Порядок расположения параметров в строке значения не имеет.

Между процедурными скобками mMDBCliProc- mMDBCliPost с помощью функций полей попарно перечисляются идентификаторы и время для интересующих ТИ.

Количество пар не ограничено. Время задается в секундах UNIX-формата, 0 – означает текущее значение.

```
void TIAppend (MDEObj CliObjDB)
{
    UINT fuExitCode = 0;
    DWORD key, dwId;
    ValDsc *Val;
    ObjDsc *Obj;
    BYTE bHour = 0;
    time_t LongTime;
    struct tm *tmStruct;
    char Prm [80];

    mMDBCliBlockBeg (CliObjDB, 0);
    strcat (Prm, "TABLE_NAME=T_ARCH_TI\r\n"); // Обязательный
    strcpy (Prm, "MYPARAM=Примечание");      // Необязательный
    mMDBCliProc (CliObjDB, "READ_ARCH", Prm);
    for (POSITION pos = TiList->GetStartPosition(); pos != NULL;)
    { TiList->GetNextAssoc (pos, key, dwId);
      mMDBCliFldSmall (CliObjDB, "ID", (short)key);
      mMDBCliFldInt (CliObjDB, "DT", 0);
    }
    mMDBCliPost (CliObjDB);
    mMDBCliBlockEnd (CliObjDB, TRUE, TRUE);
}
```

Прием данных будет происходить в той же последовательности, как это описано для SQL-запросов. Только вместо обработчика события onSqlBegin будет вызван onProcBegin, а вместо onSqlEnd - onProcEnd.

Следующий пример показывает, как изменять архивные данные: заполнение архива часовой ведомости данными ТИ. Процедура называется «WRITE_ARCH».

С помощью полей также как и в предыдущем примере, задаются идентификатор ТИ и время, за которое ТИ будет записано. Далее перечисляются значения, которые будут записаны в архив. Общее правило для формирования списка значений следующее: имена и типы полей для конкретного архива, должны соответствовать именам и типам полей, полученным в результате запуска процедуры «READ_ARCH».

```
    mMDBCliFldInt (CliObjDB, "FLG", Val -> m_ulSign);
    free (Val);
    mMDBCliBlockBeg (CliObjDB, 0);
    strcpy (Prm, "TABLE_NAME=T_ARCH_V_H_TI");
    mMDBCliProc (CliObjDB, "WRITE_ARCH", Prm);
    for (POSITION pos = TiVal -> GetStartPosition(); pos != NULL; )
    { TiVal -> GetNextAssoc ( pos, key, (ValDsc*)&Val );
      mMDBCliFldSmall (CliObjDB, "ID", (short)Val -> m_nId);
      mMDBCliFldInt (CliObjDB, "DT", 0);
      mMDBCliFldFloat (CliObjDB, "VAL", (float)Val -> m_dVal);
      TiVal -> RemoveKey (key);
    }
    mMDBCliPost (CliObjDB);
    mMDBCliBlockEnd (CliObjDB, TRUE, FALSE);
```

2.2.4.5. Изменение записей НСИ

Для изменения записей в таблицах НСИ предусмотрены три команды: mMDBCliRowNew, mMDBCliRowEdit и mMDBCliRowDel. Аналогично описанным выше командам для работы с SQL- и PROC-запросами, каждая из них должна завершаться функцией mMDBCliPost, перед которой может быть вызвано несколько команд изменения полей.

В качестве параметров, помимо идентификатора объекта, указываются имя таблицы и, для команд редактирования и удаления, WHERE-выражение, однозначно определяющее запись в таблице. Имя данного параметра - «KEY». Если в результате применения KEY-выражения будет выделено несколько записей, то команда применяется к первой из них.

```
mMDBCliBlockBeg(ObjDB, 1);

mMDBCliRowNew(ObjDB, "T_DOC", "");
mMDBCliFldText(ObjDB, "DOC_NAME", 9, "Ведомости");
mMDBCliPost(ObjDB);

mMDBCliRowEdit(ObjDB, "T_FRM", "KEY=FRM_ID=102");
mMDBCliFldText(ObjDB, "FRM_NAME", 11, "Ведомость 2");
mMDBCliFldBlob(ObjDB, "FRM_DATAE", DataLen, Data);
mMDBCliFldInt(ObjDB, "FRM_EDIT", 1);
mMDBCliPost(ObjDB);

mMDBCliRowDel(ObjDB, "T_DOC", "KEY=DOC_ID=123\r\nNONCIEVN=TRUE");
mMDBCliPost(ObjDB);

mMDBCliBlockEnd(ObjDB, TRUE, FALSE);
```

Изменение записи порождает соответствующее событие изменения НСИ. Эти события используются клиентскими АРМами и резервными серверами для оперативной поддержки целостности данных в системе.

В некоторых случаях имеет смысл совершить ряд изменений без генерации событий, а потом разово оповестить клиентов Каждое изменение в таблицах НСИ, произведенное с помощью описанных выше команд, о произведенной корректировке. С помощью ключа «NONCIEVN=TRUE», можно подавить формирование событий изменения данных НСИ.

2.2.5. Управление событиями

Обработка событий, один из важнейших элементов взаимодействия клиента с сервером ОИК. Интересующее клиента подмножество событий формируется путем задания нужных кодов событий и списка энергообъектов, с которых эти события должны поступать. События передаются клиенту немедленно после своего возникновения и поступают асинхронно по отношению к основному процессу обмена данными с сервером ОИК.

Основные этапы использования механизма событий на стороне клиента аналогичны этапам, описанным выше, для работы с данными. Создается программный объект для работы с событиями. При создании объекта с ним связывается функция-обработчик событий. С помощью функций mMDVCliConn и mMDVCliDisconn можно соответственно подключить или отключиться от обработки требуемого подмножества событий.

```
#include "mc_evn.h"
// Предварительное описание функций обработчиков событий
static void onObjNew(MDESes SesId, MDEObj ObjId);
```

```

static void onObjDel(MDEses SesId, MDEObj ObjId);
static void onObjAbort(MDEses SesId, MDEObj ObjId);

static void onEvn(MDEObj ObjId, long ObjData, char *Evn, int
FldCount, MDRFieldVal *Fld);

struct DataDsc
{
    int Work;
};

MDEObj MDV_CliObjNew(MDEses SesId)
{
    MDEObj ObjId;
    DataDsc *MyData;

    MyData = (DataDsc*)malloc(sizeof(DataDsc));
    if (!MDVcliObjNew(SesId, &onObjAbort, &ObjEvent,
        (long)MyData, &ObjId))
    {
        free(MyData);
        return 0;
    }
    return ObjId;
}

void MDV_CliObjDel(MDEObj ObjId)
{
    DataDsc *MyData = NULL;

    MDVcliObjDel(ObjId, (long*)&MyData);
    free(MyData);
}

void onEvn(MDEObj ObjId, long ObjData, char *Evn,
    int FldCount, MDRFieldVal *Fld)
{
    // . . . Обработка поступающих событий
}

```

Ниже приводится пример подключения к событиям изменения НСИ на сервере ОИК. Строка кодов событий содержит соответственно коды добавления, изменения и удаления записей НСИ. Строка энергообъектов, в данном случае, только один идентификатор – энергообъекта владельца БД. Именно этот идентификатор указывается в таблице «T_THIS».

```

mMDVcliBlockBeg(ObjEvn, 1);
mMDVcliConn(ObjEvn, "601,602,603", "1");
mMDVcliBlockEnd(ObjEvn, TRUE, FALSE);

```

2.3. ИСПОЛЬЗОВАНИЕ OLE-ИНТЕРФЕСА (Scdsys.ocx)

2.3.1. Обзор

Библиотека Scdsys.ocx в своей основе является OLE-надстройкой над технологиями OicMde.dll, которые были описаны в предыдущем разделе. Она предоставляет пользователю ряд интерфейсов, с помощью которых можно легко обеспечить взаимосвязь с сервером ОИК, используя стандартный для Windows механизм COM- и ActiveX-объектов. Приведем краткое описание объектов Scdsys.ocx.

№	Объект	Интерфейсы	Описание
1	ScadaCli	IScadaCli IscadaCliEvents	Канал связи с сервером. Дополнительно обеспечивает ряд вспомогательных свойств и методов, способствующих упрощению взаимодействия: предоставляет характеристики пользователя и главного энергообъекта, упрощает работу с системообразующими таблицами объектов (T_OBJ) и полей (T_FLD), правами доступа, синхронизирует время сервера и клиента.
2	ScadaAbo	IScadaAbo IScadaAboCmd IScadaAboEvents	Абонент. К каждому каналу, в рамках одной программы, может быть подключено несколько таких объектов. Именно через них происходит обмен данными и событиями с сервером. Каждый абонент обеспечивает абстракцию независимого, монопольного использования канала.
3	MDataRow MDARec	IMDataRow IMDARec	Виртуальные таблицы и записи в памяти. Помимо важного самостоятельного значения, таблицы могут применяться для эффективного приема больших массивов данных с сервера.
4	ModObj	IScadaObj IScadaObjEvents	Прообраз объектов используемых в оболочке АРМ - ScdArm.exe. Все объекты, которые предназначены для работы в среде АРМа, должны быть ActiveX- объектами, реализующими интерфейс IScadaObj (или его Dispatch – вариант) и обработчики событий из IScadaObjEvents.
5	ScadaMenu	IScadaMenu	С помощью этого объекта оболочка ScdArm.exe узнает о содержании меню IscadaObj-объекта
6	DataBag	IDataBag	Объект, реализующий абстракцию динамической, древовидной структуры данных. Используется для хранения сложных параметров в полях БД.
7	ScadaTim	IscadaTim IScadaTimEvents	Генератор событий таймера. Вспомогательный объект. Практическая ценность неочевидна.

Все примеры в данном разделе представлены на Delphi.

2.3.2. Канал связи с сервером (ScadaCli)

2.3.2.1. Интерфейс IScadaForm

Данный интерфейс является базовым для всех ActiveX-интерфейсов определенных в ScdSys.osx. В нем собраны свойства объекта, отвечающие за его представление как формы на экране.

Свойство	Назначение	Примечания
Visible	Видимость	Boolean. Показывает или устанавливает, является ли форма (объект) видимой
Caption	Заголовок	String. Заголовок формы (объекта)
Color	Цвет	OLE COLOR. Основной цвет формы (объекта)
Font	Шрифт	IFontDisp. Основной шрифт формы (объекта)
Active	Наличие фокуса	Boolean. Показывает, имеет ли форма (объект) фокус в данный момент
HelpFile	Файл помощи	String. Имя файла помощи
Enabled	Разрешение обработки	Boolean. Показывает или устанавливает, реагирует ли объект на события мыши, клавиатуры и таймера

2.3.2.2. Интерфейс IscadaCli

Метод	Назначение	Примечания
Open	Соединение с сервером	Перед вызовом метода предварительно должны быть заданы значения свойств SrvAddress, UserName и UserPassword. В случае успешного подключения свойство CliActive будет установлено в TRUE. Если соединения не произошло, то свойство NoUser покажет причину: неправильный адрес или характеристики пользователя.
Close	Закрытие соединения	Перед закрытием сессии, например при сохранении локальных настроек, имеет смысл отключить режим оповещения по событиям. Это делается с помощью метода CloseEvent.
TimeSynchrono	Синхронизация времени с сервером	Синхронизация всегда осуществляется неявно в момент создания сессии. Сразу после этого свойства TimeOle и TimeSec возвращают актуальное текущее время сервера соответственно в OLE- и UNIX(sec)-форматах. Преобразование между форматами можно осуществить с помощью свойств TimeOleToSec и TimeSecToOle.
ObjNameFromId ObjIdFromName	Работа с T_OBJ.	Вспомогательные функции. Определение имени объекта (поле OBJ_NAME_LAT) в таблице T_OBJ по его идентификатору (поле OBJ_ID) и наоборот. При этом в качестве дополнительной информации возвращается тип объекта (поле OBJ_OBJ_T_ID)
Lock Unlock	Управление блокировкой	Lock-Unlock вложенные директивы. При первом вызове Lock генерируется событие объекта OnLock, при последнем Unlock – событие OnUnlock. На внутреннем уровне Lock-Unlock вызываются автоматически при собственных операциях ScadaCli с сервером и обработке пользовательских запросов данных (BlockBegin-BlockEnd) выполненных с ожиданием результата.
CloseEvent	Запрет обработки сообщений	Вспомогательная функция. Может быть использована, например, перед закрытием сессии, до сохранения локальных параметров
ChangePassword	Изменение пароля	Изменяет значение пароля пользователя, требуемого при входе в систему
DataBagGet DataBagPut	Работа с таблицей T_DATA_BAG	Таблица T_DATA_BAG представляет собой хранилище произвольных данных, куда каждый может складывать все, что считает нужным. В частности, именно в ней хранятся локальные параметры и настройки объектов. Методы DataBagGet и DataBagPut автоматизируют работу с таблицей. Параметры: PrmName: string – имя параметра; IsLocal: Boolean – личный(True) или общий параметр; Value: IDataBag - Данные
Reconnect	Оптимальное подключение	При работе в системе с одним или несколькими резервными серверами, возможна балансировка нагрузки, путем равномерного распределения клиентов между ними. Метод Reconnect прозрачным образом переключает (или оставляет) клиента к оптимальному серверу. TRUE, возвращаемое методом, говорит о действительном переключении на другой резервный сервер.

Свойство	Назначение	Примечания
TimeOle TimeSec	Текущее время сервера	Текущее время сервера с точностью до sec, соответственно в OLE(VARIANT) и UNIX(long, sec) форматах.
UserPrm	Параметры пользователя	Возвращает VARIANT-значение указанного параметра для пользователя, открывшего сессию. Возможные значения параметров описаны в EUserField: ufId – идентификатор в таблице T_USRS; ufName1, ufName2, ufName3 – соответственно поля USRS_FIRST_NAME, USRS_MIDL_NAME и USRS_LAST_NAME таблицы T_USRS; ufFIO – скомпилированное имя с инициалами; ufOrg – организация (USRS_ORG); ufDep – отдел (USRS_DEP); ufPhone – телефон (USRS_PHONE); ufTab – табельный номер (USRS_TAB).
UserGrPrm	Параметры группы пользователя	Возвращает VARIANT-значение указанного параметра для группы пользователя, открывшего сессию. Возможные значения параметров описаны в EuserGrField: gfId – идентификатор в таблице T_USRGR; gfName – имя группы (USRGR_NAME); gfComm – комментарии (USRGR_COMM); gfAccess – зарезервировано.
EnergyObj	Энергообъект, владелец БД	Идентификатор энергообъекта, прописанный в таблице T_THIS
HWnd	Описатель окна	Описатель окна объекта. Используется для обмена сообщениями и подключения объектов-абонентов
AboCount	Счетчик абонентов	Количество объектов-абонентов, связанное с данным объектом ScadaCli
ObjCount ObjByIndex ObjByValue ObjIndex	Работа с таблицей T_OBJ	С помощью данных свойств можно узнать количество записей в таблице T_OBJ, получить информацию об объекте по его порядковому номеру в таблице (ObjByIndex), имени или идентификатору (ObjByValue) и, наконец, определить индекс объекта в списке по его имени или идентификатору (ObjIndex). Информация об объекте возвращается в VARIANT-массиве, элементы которого будут содержать (EObjFields): CF_Obj_Id(0) – идентификатор (OBJ_ID); CF_Obj_NameLat(1) – латинское имя (OBJ_NAME_LAT); CF_Obj_Name(2) – полное наименование (OBJ_NAME); CF_Obj_Type(3) – тип (OBJ_OBJ_T_ID); CF_Obj_ACXTBL(4) – ActiveX для таблиц (OBJ_ACXTBL); CF_Obj_ACXREC(5) – ActiveX для записей (OBJ_ACXREC); CF_Obj_ICO(6) – иконка (OBJ_ICO); CF_Obj_Hide(7) – признак скрытности (OBJ_HIDE).
FldCount FldByIndex FldByValue FldByObj	Работа с таблицей T_FLD	С помощью данных свойств можно узнать количество записей в таблице T_FLD, получить информацию о поле по его порядковому номеру в списке (FldByIndex), имени или идентификатору (FldByValue) и, наконец, получить полный список полей для заданного объекта (FldByObj). Информация о поле объекта возвращается в VARIANT-массиве, элементы которого будут содержать (EfieldFields): CF_Fld_Id(0) – идентификатор (FLD_ID); CF_Fld_Obj_Ref(1) – идентификатор объекта (FLD_OBJ_ID); CF_Fld_NameLat(2) – латинское имя (FLD_NAME_LAT);

		<p>CF_Fld_Name(3) – наименование (FLD_NAME); CF_Fld_Type_Ref(4) – тип (FLD_FLD_T_ID); CF_Fld_Flags(5) – флаги (FLD_FLAGS); CF_Fld_Size(6) – размер (FLD_SIZE); CF_Fld_Prm(7) – параметры (FLD_PRM).</p> <p>При получении полного списка полей объекта с помощью FldByObj, возвращаемое VARIANT-значение представляет собой массив массивов параметров полей, по количеству полей объекта.</p>
CliActive	Признак открытой сессии	Boolean. Сессия считается активной, если открыт канал и завершена предварительная загрузка данных объектом ScadaCli. Изменяя значение CliActive, можно открыть или закрыть сессию, аналогично использованию методов Open и Close.
SrvAddress	Адрес сервера	String. Полный сетевой адрес серверной ЭВМ или строка IP-адреса. Если задать список из 2-х и более адресов, разделенных запятыми, то попытки подключения будут производиться последовательно, слева направо, до первого успешного соединения.
UserName	Имя пользователя	String. Имя пользователя при регистрации в системе (поле USRS_NAME в таблице T_USRS).
UserPassword	Пароль	String. Пароль пользователя при регистрации в системе (поле USRS_PASS в таблице T_USRS).
NoUser	Причина неудачного подключения	Boolean. При неудачном открытии сессии, TRUE – означает неправильные имя или пароль, FALSE – имя сервера.
TimeOleToSec TimeSecToOle	Преобразование OLE- и UNIX-времени	Взаимное преобразование OLE-времени в UNIX- формат (TimeOleToSec) и, обратно, времени в секундах в OLE- время (TimeSecToOle)
DataBagId	Идентификатор записи в T_DATA_BAG	Long. Это свойство дополняет набор методов для работы с таблицей параметров - T_DATA_BAG. По имени и признаку локальный(TRUE)-общий(FALSE), ищется запись и возвращается ее идентификатор. Если запись не найдена, возвращается нулевое значение.
WorkDir	Рабочий директорий	String. Имя рабочего директория. Чисто информационное свойство. Внутри ScadaCli не используется.
DragData	Хранилище данных при DragDrop операциях	IDataBag. Помимо стандартных OLE-механизмов, используемых в DragDrop-операциях, можно использовать данное свойство как промежуточное хранилище данных. Внутри ScadaCli не используется.
ClipData	Хранилище данных при операциях с буфером обмена	IDataBag. Помимо стандартных OLE-механизмов, используемых при операциях с буфером обмена, можно использовать данное свойство как промежуточное хранилище данных. Внутри ScadaCli не используется.
RightObj RightProc	Права доступа	Int. Возвращает значения прав доступа для данного пользователя соответственно из таблиц T_OBJ_R и T_PROC_R. Объекты и процедуры могут быть указаны как по имени, так и по идентификатору в таблице. Биты прав доступа для объекта следующие (EobjRights): 0 – orRead, 1 – orWrite, 2 – orNew, 3 – orDel;
ReconnectAuto	Автоматическое переподключение	Boolean. Если данное свойство установлено в TRUE, то при аварийном разрыве связи, объект ScadaCli будет пытаться повторно подключиться к одному из серверов указанных в списке SrvAddress. Данное подключение будет осуществлено прозрачно для пользователя с полным сохранением текущего контекста.

InProcess	Признак занятости	Boolean. Значение равно TRUE, если CliActive = TRUE и объект ScadaCli, либо один из абонентов находятся в состоянии обработки события или формирования блока запроса. Иначе – FALSE.
-----------	-------------------	--

2.3.2.3. События объекта

Событие	Назначение	Примечания
OnSesClose	Закрытие сессии сервером	Возникает в следствии завершения сессии сервером ОИК
OnSesAbort	Аварийный разрыв связи	Аварийное завершение сессии. Например, повреждение сети, выключение ЭВМ сервера и т.п.
OnSesUser	Изменение характеристик пользователя	Изменение характеристик пользователя на сервере (имя, пароль и т.п.). Новые характеристики доступны через свойство UserPrm.
OnLock OnUnlock	Обработка блокировки	Следствие использования методов Lock-Unlock.

2.3.2.4. Установка и разрыв соединения

```

function ScdConnect(Cli: TScadaCli): Boolean;
begin
    Cli.UserName := 'Ковтун';
    Cli.UserPassword := '1';
    Cli.SrvAddress := 'Oic1';
    Cli.Open;
    if not Cli.CliActive then
        if Cli.NoUser then
            Application.MessageBox('Имя или пароль неверны',
                'Вход в систему', MB_OK + MB_ICONSTOP)
        else
            Application.MessageBox('Сервер не найден',
                'Вход в систему', MB_OK + MB_ICONSTOP);
        Cli.WorkDir := 'D:\Scd\Temp';
        Result := Cli.CliActive;
    end;
    ScadaCli.Open;
    Result := ScadaCli.CliActive

FScadaCli := TScadaCli.Create(Self);
with FScadaCli do begin
    Name := 'ScadaCli';
    Parent := Self;
    Visible := False;
    OnSesAbort := ScadaCliSesAbort;
    OnSesClose := ScadaCliSesClose;
    OnSesUser := ScadaCliSesUser;
    OnLock := ScadaCliLock;
    OnUnlock := ScadaCliUnlock;
    ReconnectAuto := FConnBest;
end;

```


2.3.3. Абонент (ScadaAbo)

2.3.3.1. Интерфейс IDataRec

Данный интерфейс используется для доступа к записи данных в обработчиках событий абонента OnRowValue и OnNotify. Запись данных состоит из одного или нескольких полей (FieldsCount). Каждое поле имеет имя, содержит данные определенного типа. Доступ к полям может производиться по индексу или по имени поля.

Метод	Назначение	Примечания
GetValue	Получение значения и параметров поля	GetValue(Field: OleVariant; out AType: Integer; out ASize: Integer): OleVariant; Метод возвращает все сразу: значение, тип и размер данных в байтах. Поле можно указать по индексу или его имени.
Свойство	Назначение	Примечания
FieldsCount	Количество полей в записи	Integer
FieldName	Имя поля	FieldName[Index: Integer]: String; Получение имени поля по индексу
FieldIndex	Индекс поля	FieldIndex[const Name: String]: Integer; Определение индекса поля по его имени. Если поля с таким именем не найдено, возвращается -1;
FieldValue	Значение поля	FieldValue[Field: Variant]: Variant;
FieldType	Тип поля	FieldType[Field: Variant]: EFieldType;
FieldSize	Размер поля	FieldSize[Field: Variant]: Integer;

2.3.3.2. Интерфейс IScadaAbo

Основной интерфейс, через который осуществляется работа с данными сервера: HCI, архивами и событиями.

Метод	Назначение	Примечания
BlockBegin BlockEnd	Формирование блочного запроса	Все запросы к серверу на чтение, изменение данных или запуск серверных процедур выполняются в рамках блока. При выполнении команды BlockEnd можно отказаться от изменений, сделанных в БД (IsCommit=FALSE) и, ожидать результатов запроса (IsWait=TRUE) или продолжить выполнение без ожидания (IsWait=FALSE)
Proc	Запуск серверной процедуры	Proc(TblHandle: Integer; ReplaceEq: WordBool; DoNotify: WordBool); Пример использования. Чтение архива: with ScadaAbo. do begin RequestPrm['PROC']:='READ_ARCH'; RequestPrm['TABLE_NAME']:='T_ARCH_TI'; Proc(0,False,False); FieldValue('ID',eftInt,101); FieldValue('DT',eftInt,0); // Текущее Post; end; Три параметра, используемые в процедуре, связаны возможностью использования виртуальных таблиц для приема данных запроса. Если создать пустой объект MDArray и указать MDArray.Handle в качестве первого параметра, то данные будут приниматься в эту таблицу. Параметр ReplaceEq управляет режимом обработки

		<p>совпадающих (по ключу) записей. Если DoNotify=FALSE, то, при наличии таблицы, обработчики OnProcBegin и OnProcEnd вызываться не будут.</p> <p>Если используемая виртуальная таблица не имеет структуры, то при приеме данных, автоматически будет сформирован список полей, соответствующий полям принимаемой записи. Если же список полей уже имеется в наличии, то при приеме данных будут использоваться уже существующие поля и индексы.</p>
Sql	Выполнение SQL-запроса	<p>Sql(TblHandle: Integer; ReplaceEq: WordBool; DoNotify: WordBool);</p> <p>Пример использования. Чтение таблицы T_TI:</p> <pre>with ScadaAbo. do begin RequestPrm['SQL']:='SELECT * FROM T_TI'; RequestPrm['NAME']:='T_TI'; Sql(0,False,False); Post; end;</pre> <p>Назначение параметров SQL-метода полностью совпадает с параметрами, описанными для Proc;</p>
RowNew	Добавление новой строки НСИ	<p>Пример использования. Добавление в T_RTS:</p> <pre>with ScadaAbo. do begin RequestPrm['TABLE']:='T_RTS'; RowNew; FieldValue('RTS_USER_ID',eftInt,13); FieldValue('RTS_NAME',eftChar,'Тест'); // ... Post; end;</pre>
RowEdit	Изменение строки НСИ	<p>Пример использования. Изменение имени в T_DOC:</p> <pre>with ScadaAbo. do begin RequestPrm['TABLE']:='T_DOC'; RequestPrm['KEY']:='DOC_ID=35'; RowEdit; FieldValue('DOC_NAME',eftMemo,'Док2'); Post; end;</pre>
RowDelete	Удаление строки НСИ	<p>Пример использования. Удаление схемы в T_SCH:</p> <pre>with ScadaAbo. do begin RequestPrm['TABLE']:='T_SCH'; RequestPrm['KEY']:='SCH_ID=10'; RowDelete; Post; end;</pre>
Post	Завершение команд	<p>Все команды блока (Proc, Sql, RowNew, RowEdit, RowDelete) должны завершаться командой Post. Между ними, при необходимости, могут быть заданы значения полей с помощью FieldValue.</p>
FieldValue	Задание значения поля	<p>Параметрами являются <имя поля>, <тип> и <присваиваемое значение>. Возможные типы поля описаны в EfieldType (eftBool, eftByte, eftSmall, eftInt, eftFloat, eftDouble, eftChar, eftMemo, eftBlob, eftUnixDT, eftUnixMDT, eftCurrency)</p>
EventConnect	Подключение к оповещению по событиям	<p>Параметры EventConnect – два variant-массива. Первый – список кодов требуемых кодов событий, второй – список идентификаторов энергообъектов, на которых эти события</p>

		<p>интересуют пользователя-абонента. События изменения НСИ для энергообъекта – владельца БД всегда принимаются по умолчанию. Повторный вызов EventConnect обнуляет старый список оповещения для данного абонента и заменяет его на новый.</p> <pre>With ScadaAbo. do begin Cod:=VarArrayCreate([0,1],varInteger); Cod[0]:=101; Cod[1]:=102; Eno:=VarArrayCreate([0,2],varInteger); Eno[0]:=7; Eno[1]:=10; Eno[2]:=13; EventConnect(Cod, Eno); end;</pre> <p>Если в списке встречается «0», то это означает - для всех элементов (энергообъектов или кодов событий)</p>
FieldDef	Параметры поля таблицы-результата	<p>В обработчиках событий OnSqlBegin и OnProcBegin можно узнать структуру полей записи с помощью FieldDef. Свойство FieldsCount возвращает количество полей в записи. FieldDef для каждого поля (по индексу) выдает соответственно: тип, размер и наименование.</p>
Свойство	Назначение	Примечания
RequestPrm	Задание параметров запроса	<p>Параметром запроса называется пара <Имя>=<Значение>. Совокупность таких параметров для каждой команды передается на сервер и используется им для конкретизации действий.</p> <p>Все параметры, в том числе и те, которые сервером не используются, возвращаются с результатом запроса и становятся доступны в обработчиках OnSqlBegin и OnProcBegin с помощью свойства OnAnswerPrm.</p> <p>Примеры применения RequestPrm см. выше, в примерах команд Proc, Sql и т.п.</p>
OnAnswerPrm	Параметры результата запроса	<p>С помощью свойства OnAnswerPrm можно при получении результата запроса узнать значения параметров, переданные на сервер командами Proc и Sql.</p> <p>Все параметры, в том числе и те, которые сервером не используются, возвращаются с результатом запроса и становятся доступны в обработчиках OnSqlBegin и OnProcBegin.</p>
CliHWnd	Описатель окна объекта ScadaCli	<p>С помощью данного свойства объект ScadaAbo связывается с объектом ScadaCli. Пример:</p> <pre>ScadaAbo.CliHWnd:=ScadaCli.hWnd;</pre>
ScadaCli	Интерфейс IscadaCli	<p>IscadaCli. Если с данным абонентом не связан объект ScadaCli, то NIL.</p>
LastErrorCode LastErrorStr	Код и строка последней ошибки	<p>Код и строка последней ошибки, возникшей в результате обработки блока данных на сервере. Обычно, после возникновения такой ошибки, изменения БД, сделанные в данном блоке –откатываются, генерируется сообщение об ошибке, дальнейшее выполнение команд блока прекращается.</p>
FieldsCount	Количество полей в записи результата запроса	<p>Актуально в теле обработчиков OnProcBegin и OnSqlBegin. Характеристики полей можно получить по индексу с помощью метода FieldDef.</p>
AboActive	Рабочее состояние абонента	<p>Абонент ScadaAbo подключен к ScadaCli и ScadaCli.CliActive=TRUE</p>
InRequestBlock	Абонент	<p>Объект абонента находится в процессе подготовки блока</p>

	подготавливает блок запроса	запроса, т.е. BlockBegin-BlockEnd;
InAnswerBlock	Абонент принимает данные	Объект абонента находится в процессе приема и обработки данных, полученных в результате запроса к серверу.

2.3.3.3. События объекта

Большинство обработчиков событий объекта, связано с процессом приема и обработки данных, полученных в результате запроса к серверу. Последовательность вызова этих процедур следующая:

```
OnBlockBegin;
  OnProcBegin(var TblHandle: Integer; var ReplaceEq:
WordBool); //OnSqlBegin;
  OnRowValue(const RecData: IDataRec; IsSql: WordBool);
  OnProcEnd(TblHandle: Integer); //OnSqlEnd
  OnError(ErrCode: Integer; const ErrStr: WideString);
OnBlockEnd(IsCommit: Boolean);
```

Каждый объект абонента, всегда подключен к оповещению о событиях изменения НСИ, возникающих на сервере, при корректировке таблиц НСИ. При необходимости, подключение к другим событиям, осуществляется с помощью метода EventConnect. В любом случае, процесс обработки сообщений использует единственный обработчик:

```
OnNotify(EventCode: Integer; const RecData: IDataRec);
```

Событие	Назначение	Примечания
OnBlockBegin OnBlockEnd	Начало и конец блока результата	Обработчики OnBlockBegin и OnBlockEnd вызываются всегда, для каждого блока команд, выполненного данным абонентом на сервере.
OnProcBegin OnProcEnd	Начало и конец результата процедуры	В обработчике OnProcBegin можно установить (или изменить) виртуальную таблицу или ее параметры. Параметры, сформированные перед запуском процедуры Proc, доступны через свойство OnAnswerPrm. Если в команде «Proc» была задана виртуальная таблица для принятия данных и, параметр DoNotify был TRUE, то OnProcBegin и OnProcEnd .вызываться не будут.
OnSqlBegin OnSqlEnd	Начало и конец результата SQL-запроса	Аналогично обработке OnProcBegin-OnProcEnd.
OnRowValue	Запись (строка) результата	Вызывается для каждой записи результата Proc- или Sql-запроса. Сколько записей – столько вызовов. Если задана виртуальная таблица, то данные складываются непосредственно в нее и OnRowValue не вызывается.
OnError	Ошибка	Если в процессе обработки блока запроса на сервере произошла ошибка, то дальнейшее выполнение команд блока прекращается, произведенные изменения «откатываются», генерируется данное событие с кодом и строкой ошибки и блок завершается
OnNotify	Запись (строка) события	OnNotify(EventCode: Integer; const RecData: IDataRec); Если абонент подключен к оповещению по событиям, то данный обработчик будет вызван для каждого такого события, зарегистрированного на сервере.
OnAboActive	Изменение режима работы	Вызывается, если изменяется свойство абонента AboActive.

2.3.4. Дополнительные возможности

2.3.4.1. Таблицы в памяти. IMDArray, IMDARec

Таблицы в памяти (MDAarray) представляют собой набор однотипных записей (MDARec), связанных между собой различными отношениями порядка.

Формирование и просмотр структуры записи (полей) осуществляется с помощью методов и свойств: FieldAdd, FieldGet, FieldCount.

Работа с индексами (ключами), задающими порядок записей в таблице: KeyAdd, KeyDel, KeyGet, ClearKey, KeyCount.

Создание, удаление и редактирование записей: RecNew, RecEdit, RecPost, RecDel, ClearRec, RecCount.

Сортировка, поиск и прямой доступ к записям: Sort, Seek, Rec, RecIdx.

Метод ClearAll, полностью освобождает все данные, связанные с таблицей.

Метод	Назначение	Примечания
FieldAdd	Добавление нового поля	FieldAdd(const AName: String; AType: EFieldType; ALen: Integer). Применяется при первоначальном формировании структуры записи. Имена полей не должны совпадать. Параметр длины имеет смысл только для строкового типа фиксированной длины. При изменении структуры полей все существующие записи данных уничтожаются.
FieldGet	Получение характеристик поля	FieldGet(AFld: Variant; out ANamIdx: Variant; out AType: EFieldType; out ASize: Integer); Обращение к полю можно осуществлять как по имени, так и по индексу. При этом, в зависимости от выбранного метода, в параметре ANamIdx, будет возвращено альтернативное значение.
KeyAdd	Добавление ключа (индекса)	KeyAdd(const AFields: String; AOptions: Integer); С помощью ключей задается порядок записей в наборе. Ключ определяется набором имен полей (параметр AFields), перечисленных через «;» и дополнительными опциями (AOptions). Значение Aoptions, комбинация флагов AOptions: IdxPrimary(1)-первичный ключ; IdxUnique(2)-уникальный, не может быть записей с совпадающим ключом; IdxDescending(3)-упорядочить по убыванию (порядок по умолчанию - возрастание); IdxCaseInsensitive(4)-не учитывать регистр при сравнении строк; IdxExpression(5)-(зарезервировано); IdxNonMaintained(6)-индекс не корректируется, при изменении записей. Перестройка индекса должна осуществляться явно командой Sort.
KeyDel	Удаление ключа	KeyDel(AKey: Integer) Невозможно удалить последний ключ, если в таблице есть записи.
KeyGet	Получение характеристик ключа	KeyGet(AKey: Integer; out AFields: String; out AOptions: Integer);
Copy	Копирование из другой таблицы	Copy(const ASrc: IMDArray; ARec: WordBool; AKey: WordBool) Копирование структуры полей другой таблицы. Если ARec или AKey = TRUE, то будут также скопированы все записи и (или) ключи таблицы-источника.

ClearRec	Удаление всех записей	Структура полей и ключи остаются
ClearKey	Удаление ключей	Структура полей и записи остаются. Если в таблице есть записи, то останется один ключ с пустым набором индексных полей.
ClearAll	Очистка таблицы	Удаляются все записи, все ключи таблицы и, наконец, очищается список полей.
RecNew	Создание новой записи	<p>RecNew(const ATarget: IMDARec): IMDARec;</p> <p>Принцип добавления новой записи следующий: командой RecNew создается пустая запись, устанавливаются значения ее полей, командой RecPost, запись размещается в таблице. Если объект ATarget = nil, то будет создан и возвращен новый объект IMDARec, если же ATarget <> nil, то новая запись будет связана с ATarget и RecNew вернет его значение. Пример:</p> <pre> ATbl: IMDArray; ARec: IMDARec; ARec:=ATbl.RecNew(nil); ARec.Value['ID']:=177; ARec.Value['NAME']:=‘ТИ’; ATbl.RecPost(ARec); </pre> <p>Для отказа от добавления уже созданной записи, нужно применить к ней метод RecDel.</p>
RecEdit	Изменение существующей записи	<p>RecEdit(const ARec: IMDARec; const ATarget: IMDARec): IMDARec;</p> <p>Данный метод создает копию редактируемой записи и возвращает ее в качестве результата. После внесения изменений, метод RecPost заменит исходную запись на ее измененный вариант.</p> <p>Для отказа от фиксации изменений нужно применить к записи возвращенной RecEdit метод RecDel.</p>
RecPost	Фиксация изменений	<p>RecPost(const ARec: IMDARec);</p> <p>Фиксирует изменения начатые с помощью команд RecNew или RecEdit.</p>
RecDel	Удаление записи	<p>RecDel(const Arec: IMDARec);</p> <p>Удаляет запись из таблицы. Если метод применяется к указателям на записи, полученным с помощью RecNew или RecEdit, то это означает отказ от изменений.</p>
Seek	Поиск записи	<p>Seek(AKey: Integer; AKeyVal: Variant; out ARecIdx: Integer; const ATarget: IMDARec): IMDARec;</p> <p>Поиск записи осуществляется по ключу, индекс которого задается в AKey. AKeyVal – искомое значения, в соответствии со списком полей для данного ключа. Когда полей несколько, то AKeyVal – массив Variant-ов. Если запись найдена, то Seek возвращает указатель на нее (или ATarget, если он был задан). Иначе возвращается nil. В любом случае ARecIdx – индекс найденной или первой большей записи из списка для данного ключа.</p>
Sort	Сортировка по ключу	<p>Sort(AKey: Integer)</p> <p>Принудительная сортировка. Имеет смысл для ключей со свойством IdxNonMaintained.</p>
Rec	Запись из списка	<p>Rec(AKey: Integer; AIdx: Integer; const ATarget: IMDARec): IMDARec;</p> <p>Для каждого ключа записи упорядочены по разному. Поэтому для получения записи нужно указать ключ и индекс ее в списке.</p>

Свойство	Назначение	Примечания
FieldCount	Количество полей	Integer
KeyCount	Количество ключей	Integer
RecIdx	Индекс записи в списке	RecIdx[AKey: Integer; const ARec: IMDARec]: Integer; Для любого ключа можно определить индекс записи в списке.
RecCount	Количество записей	Integer
LastError	Ошибка	EADArrayErr Если какая нибудь операция закончилась неудачно, то тип последней ошибки можно узнать с помощью данного свойства: RAE_OK – все нормально; RAE_NOMEM – недостаточно ОП; RAE_TRANSLATE – возникает при добавлении полей или ключей. Например, имя поля уже существует или, поле указанное в ключе, отсутствует в списке полей; RAE_NOFIELDS – поля в таблице не определены; RAE_OTHERFIELDS – попытка утвердить изменения если запись и таблица разной структуры; RAE_DATAREC – структура данных записи нарушена; RAE_NOREC – при выполнении метода RecPost, не найдена исходная, изменяемая запись; RAE_KEYUNIQUE – запись не может быть добавлена или изменена, т.к. нарушается уникальность ключа.
RecChanges	Количество изменений	Integer Данное поле можно обнулять. Каждое изменение записей наращивает значение RecChanges
Handle	Описатель	Integer
Tag	Поле пользователя	Integer

Запись, это собственно та структура, которая хранит сами данные. Некоторые ее методы и свойства дублируют аналогичные в таблице: FieldGet, FieldCount.

Работа со значениями полей осуществляется с помощью: Value, AsText, IsNull.

Метод	Назначение	Примечания
FieldGet	Получение характеристик поля	FieldGet(AField: Variant; out AnamIdx: Variant; out AType: EFieldType; out ASize: Integer); Аналогично FieldGet для IMDArray
Свойство	Назначение	Примечания
Handle	Описатель записи	Integer
AsText	Значение поля как текст	AsText[AField: Variant]: String; Текстовое значение поля. Поле может быть указано как по имени, так и по индексу в списке полей.
Value	Значение поля	Value[AField: Variant]: Variant;
IsNull	Признак пустого поля	IsNull[AField: Variant]: WordBool;
IsEdit	Признак измененного значения	IsEdit[AField: Variant]: WordBool;
ArrHandle	Описатель таблицы	Integer
FieldCount	Количество полей	Integer Аналогично FieldCount для IMDArray

2.3.4.2. Хранилище данных. IDataBag

Объекты, реализующие интерфейс IDataBag, используются в системе для работы с данными древовидной структуры, похожей на структуру каталогов Windows.

«Дерево» состоит из поименованных узлов, каждый из которых имеет VARIANT-значение и может быть владельцем произвольного количества других, подчиненных ему узлов. Списки подчиненных узлов упорядочены по именам. Имена внутри одного списка должны быть уникальны. Всегда существует главный, корневой узел всего «дерева».

В большинстве методов и свойств интерфейса используется параметр «Path» (путь), с помощью которого указывается узел, к которому будет применено действие. Путь – это список имен и (или) индексов, по которому можно последовательно добраться до искомого узла, начиная с корневого. Корневой узел имени не имеет и идентифицируется пустой строкой или NULL-значением Path.

Для удобства работы можно связать временное положение корневого узла с любым существующим узлом в дереве. Это делается с помощью свойства RootPath; При этом свойство RootIsEmpty будет показывать, совпадает ли виртуальный корневой узел с реальным.

В качестве VARIANT-значений узлов могут быть использованы массивы.

Метод	Назначение	Примечания
SetNode	Присвоение значения узлу	SetNode(Path: Variant; Val: Variant): WordBool; Если узлы пути не найдены, но указаны по именам, то они создаются. Последнему узлу присваивается значение Val. Если значение присвоено, метод возвращает TRUE;
DelNode	Удаление узла	DelNode(Path: Variant); Удаление узла приводит к удалению его значения и всех подчиненных ему узлов дерева. Нельзя удалить корневой узел.
Clear	Очистка корневого узла	Удаляются данные и все подчиненные узлы для корневого узла

Свойство	Назначение	Примечания
Name	Имя узла	Name[Path: Variant]: String; Получение имени узла. Пустая строка, если узел не найден
Index	Индекс узла в списке подчиненных	Index[Path: Variant]: Integer; Получение индекса узла в списке подчиненных узлов которому он принадлежит. -1, если не найден
ChildCount	Количество подчиненных узлов	ChildCount[Path: Variant]: Integer; Количество узлов непосредственно подчиненных указанному
Data	Значение узла	Data[Path: Variant]: Variant;
RootPath	Реальный путь к виртуальному корневому узлу	RootPath: Variant; Путь NULL или пустая строка соответствует реальному корневому узлу дерева
Block	Данные корневого и подчиненных ему узлов	String Данные корневого и подчиненных ему узлов, упакованные в строку
BagTree	Данные корневого и подчиненных ему узлов	BagTree[Path: Variant]: IDatBag; Данные корневого и подчиненных ему узлов, скопированные в новый объект
RootIsEmpty	Признак совпадения реального и виртуального корня	WordBool

3. АРМ ПОЛЬЗОВАТЕЛЯ

3.1. НАРАЩИВАНИЕ ФУНКЦИОНАЛЬНОСТИ

3.1.1. Обзор

Программа АРМ пользователя (ScdArm.exe) по своей архитектуре является типичным контейнером ActiveX-объектов. Она обеспечивает загрузку и поддержку функциональных модулей системы, контроль и обновление версий используемых файлов и библиотек, отвечает за общий сценарий работы с сервером ОИК.

Однако, сами функциональные модули, не являются частью программы АРМ. Они представляют собой библиотеки ActiveX-объектов, которые могут быть разработаны независимо, на любых языках, поддерживающих OLE-технологии Microsoft.

Для того, чтобы ActiveX-объект мог использоваться в составе оболочки АРМ, он должен реализовать интерфейс IscadaObj или его Dispatch-вариант. Данный интерфейс, наряду с многими другими, определен в ScdSys.ocx.

3.1.2. Интерфейс IscdDisp

Данный интерфейс предназначен для взаимодействия с произвольной объектной моделью из языка Tscript.

Свойство	Назначение	Примечания
Prop	Доступ к свойствам объекта	Prop([in] VARIANT AName) VARIANT Prop([in] VARIANT AName, [in] VARIANT Value); AName – имя или идентификатор свойства. Если свойство индексированное, то AName – массив, первый элемент которого имя или идентификатор свойства;
Метод	Назначение	Примечания
Call	Выполнение метода объекта	Call([in] VARIANT AName, [in] VARIANT Param): VARIANT; AName – имя или идентификатор метода; Param – параметр или массив параметров.
Событие	Назначение	Примечания
OnEvent	Событие	OnEvent([in] VARIANT Name, [in] VARIANT Param); Оповещение о событиях объекта. Name – имя или идентификатор события; Param – дополнительные параметры;

3.1.3. Интерфейс IscadaObj

Метод	Назначение	Примечания
ObjDataCancel	Отказаться от изменений	Данный метод дает директиву объекту – отказаться от сделанных изменений данных, если они есть. О наличии таких изменений можно судить по значению свойства ObjDataIsEdit, а обработчик событий OnObjData должен информировать об изменении состояния данных.
ObjDataSave	Сохранить данные	Директива – сохранить измененные данные.
ObjStateLoad ObjStateSave	Загрузка-сохранение состояния в БД	ObjStateLoad(Ench: WordBool); ObjStateSave(Ench: WordBool); Директива – загрузить (сохранить) текущее состояние в БД. Ench – признак «расширенного» набора свойств. Обычно, объекты хранят состояние в таблице

		<p>T_DATA_BAG, используя методы ScadaCli. DataBagGet и ScadaCli.DataBagPut. Например:</p> <pre> procedure TModObj.ObjStateLoad(Ench: WordBool); var IB: IDataBag; begin IB := ScadaCli.DataBagGet('STATE_' + Copy(ClassName,2,50), True); InternalStateSet(Ench, IB); end; procedure TModObj.ObjStateSave(Ench: WordBool); var IB: IDataBag; begin IB := InternalStateGet(Ench); ScadaCli.DataBagPut('STATE_' + Copy(ClassName, 2, 50), True, IB); end; </pre>
ObjProcessStart ObjProcessStop	Начало и конец работы	Начало и конец работы модуля. В начале работы модуль считывает данные с сервера и инициализирует свои структуры данных. В конце – все данные очищаются. Если модуль активен, то ObjInProgress = TRUE
ObjParamGet ObjParamPut	Чтение-установка параметров	ObjParamGet(Param: Variant): Variant; ObjParamPut(Param: Variant; Value: Variant); Объект может не поддерживать работу с параметрами, однако работа с параметрами локального состояния желательна: 'STATE' и 'STATEENCH'. Это те же самые данные, которые сохраняются и загружаются в ObjStateLoad и ObjStateSave.
Свойство	Назначение	Примечания
ObjCliHWnd	Описатель окна объекта ScadaCli	Обычно, для работы с данными сервера, сразу после своего создания, объект создает своего абонента - ScadaAbo. Первое, что делает АРМ, создав объект, это передает ему описатель канала через это свойство. С его помощью ActiveX подключает абонента к каналу.
ObjMenu	Меню	Одна из функций АРМ, отображение меню активного объекта. Для этого он использует данное свойство. Подробное описание интерфейса IscadaMenu см. ниже в следующем разделе
ObjDataIsEdit	Признак наличия изменений	Данное свойство показывает наличие в объекте не сохраненных данных.
ObjCliActive	Канал активен	TRUE, обычно означает, что у объекта есть объект ScadaAbo, он подключен к объекту канала (ScadaCli) и данный канал активен.
ObjInProgress	Процесс активен	Объект находится в процессе работы: общается с сервером, обрабатывает сообщения мыши и клавиатуры и т.п. Дело в том, сто функциональный модуль может быть подключен к каналу, но бездействовать. Начало и конец процесса обработки должны регламентироваться директивами ObjProcessStart и ObjProcessStop
ObjInNetEvent	Признак обработки сообщения	Если объект занят (например, принимает или обрабатывает данные) и на данный момент не расположен выполнять других директив, то данный признак может сигнализировать об этом

3.1.4. События объекта

Механизм событий используется для оповещения контейнера об изменении состояний объекта, чтобы он мог адекватно на них реагировать.

Событие	Назначение	Примечания
OnObjCaption	Изменение заголовка	Произошло изменение заголовка (свойства Caption) объекта
OnObjMenu	Изменение меню	Произошло изменение меню (свойство ObjMenu) объекта
OnObjData	Изменение данных	Изменилось состояние данных. Данные были скорректированы, но еще не сохранены (свойство ObjDataIsEdit)
OnObjNotify	Оповещение контейнера	OnObjNotify(var Param: OleVariant); С помощью данного события объект может оповещать свой контейнер о чем угодно. Главное, чтобы контейнер понимал, чего от него хотят.

3.1.5. Меню. IscadaMenu

Свойство	Назначение	Примечания
Caption	Текст меню	String
ShortCut	Горячая клавиша	Integer. Пример формирования кода (Menus.pas): Function ShortCut(Key: Word; Shift: TShiftState): TShortCut; begin Result := 0; if WordRec(Key).Hi <> 0 then Exit; Result := Key; if ssShift in Shift then Inc(Result, scShift); if ssCtrl in Shift then Inc(Result, scCtrl); if ssAlt in Shift then Inc(Result, scAlt); end;
HelpContext	Идентификатор контекстной помощи	Integer.
Checked	Отмеченный	Boolean
RadioItem	Группа	Boolean
Enabled		Boolean
Visible		Boolean
Bitmap		
GroupIndex		Integer
Name	Имя	String
Tag	Доп. значение	Integer
ItemNext	Следующий в списке	IscadaMenu
ItemChild	Первый подчиненный	IscadaMenu