



Руководство по интеграции ПК Интеллект (HTTP API, IIDK, ActiveX)

1. Руководство по интеграции ПК Интеллект. Введение. . . . .	5
2. Интеграция аппаратно-программных модулей . . . . .	5
2.1 INTELLECT INTEGRATION DEVELOPER KIT (IIDK) . . . . .	5
2.1.1 Общие сведения об IIDK . . . . .	5
2.1.1.1 Назначение IIDK . . . . .	5
2.1.1.2 Требования к разработчику . . . . .	6
2.1.1.3 Состав IIDK . . . . .	6
2.1.2 Подключение к ПК Интеллект . . . . .	6
2.1.2.1 Параметры подключения . . . . .	6
2.1.2.2 Объект Интерфейс IIDK . . . . .	7
2.1.2.3 Настройка пересылки событий через объект Интерфейс IIDK . . . . .	7
2.1.2.4 Особенности интеграции банкоматов. Объект Банкомат . . . . .	8
2.1.3 Функции IIDK . . . . .	9
2.1.3.1 Connect . . . . .	9
2.1.3.2 SendMsg . . . . .	10
2.1.3.3 Disconnect . . . . .	11
2.1.3.4 Другие функции . . . . .	11
2.1.4 Синтаксис отправляемых сообщений . . . . .	14
2.1.4.1 Синтаксис сообщений . . . . .	15
2.1.4.2 Синтаксис сообщений (900 порт) . . . . .	15
2.1.4.3 Использование классов Event и React . . . . .	17
2.1.5 Примеры управления объектами системы . . . . .	17
2.1.5.1 Добавление, изменение и удаление объектов системы . . . . .	17
2.1.5.2 Особенности работы с системой в многопользовательском режиме . . . . .	18
2.1.5.3 Определение компьютера, на котором был выгружен ПК Интеллект (через 1030 порт) . . . . .	18
2.1.5.4 Вывод видеокамеры на монитор . . . . .	18
2.1.5.5 Получение параметров объекта (через 1030 порт). GET_CONFIG . . . . .	18
2.1.5.6 Получение информации о состоянии объекта. GET_STATE и GET_LIST . . . . .	19
2.1.5.7 Вывод информационного сообщения. SET_STATE . . . . .	20
2.1.5.8 Работа с живым и архивным видео . . . . .	20
2.1.5.9 Управление телеметрией . . . . .	21
2.1.5.10 Операции со слоем карты . . . . .	21
2.2 Интеграция аппаратно-программных модулей с ПК Интеллект . . . . .	21
2.2.1 Общие сведения об интеграции аппаратно-программных модулей . . . . .	21
2.2.2 Редактирование DBI-файла . . . . .	22
2.2.2.1 Добавление объектов в intellect.dbi . . . . .	22
2.2.2.2 Использование утилиты ddi.exe для работы с DBI-файлами . . . . .	26
2.2.3 Редактирование DDI-файла . . . . .	28
2.2.3.1 Добавление в intellect.ddi информации об объекте . . . . .	28

2.2.3.2	Использование утилиты ddi.exe для работы с DDI-файлами	31
2.2.4	Дополнительные возможности утилиты ddi.exe	35
2.2.5	Разработка MDL-файла	36
2.2.5.1	Мастер создания MDL-файла	45
2.2.6	Разработка RUN-файла	46
2.2.7	Создание и настройка интегрированных объектов (модулей) в ПК Интеллект	47
3.	Элемент управления ActiveX CamMonitor.ocx	50
3.1	Общее описание ActiveX-компонента CamMonitor.ocx	50
3.2	Установка CamMonitor.ocx	51
3.3	Параметры CamMonitor.ocx	51
3.4	Методы CamMonitor.ocx	54
3.5	События CamMonitor.ocx	57
4.	HTTP API ПК Интеллект	57
4.1	Общие сведения о HTTP API	58
4.2	Версия продукта	58
4.3	Карта	58
4.3.1	Получение списка карт	59
4.3.2	Информация об одной карте	59
4.3.3	Список слоёв для выбранной карты	60
4.3.4	Информация о конкретном слое	61
4.3.5	Фоновый рисунок слоя	61
4.3.6	Список точек на слое	61
4.3.7	Информация об отдельной точке на слое	64
4.4	Классы объектов	64
4.4.1	Список классов объектов, которые существуют на сервере	64
4.4.2	Отдельный класс объектов	65
4.4.3	Список состояний для определённого класса объектов	65
4.4.4	Информация о конкретном состоянии	65
4.4.5	Получение иконки для определённого состояния	66
4.5	Объекты	66
4.5.1	Получение списка объектов	66
4.5.2	Информация об отдельном объекте	68
4.5.3	Состояние отдельного объекта	68
4.5.4	Список доступных действий с объектом, находящимся в определённом состоянии	69
4.6	Получение событий	70
4.6.1	Получение событий видеоподсистемы блоками	72
4.7	Отсылка команд на сервер	73
4.8	Макрокоманды	74
4.9	Видео	75

4.9.1 Запрос миниатюр (скриншотов)	75
4.9.2 Запрос конфигурации	76
4.9.3 Запрос видео	78
4.9.4 Формат основного потока	78
4.9.4.1 Управление записью	80
4.9.4.2 Постановка и снятие с охраны камеры	80
4.9.4.3 Управление телеметрией	80
4.9.4.4 Работа с архивом	81
4.10 Нотификация	86
4.11 Звук	88
4.11.1 Получение живого звука	88
4.11.2 Проигрывание звука из архива	89
4.11.3 Отправка живого звука	90
4.12 Команды, используемые для интеграции ЕЦХД	90
4.12.1 Список камер и их параметры	90
4.12.2 Диапазоны доступных архивных записей	92
4.12.3 Работа с видеопотоками	93
4.12.4 Выгрузка архивов	93
4.12.5 Управление функциями средства видеонаблюдения	94
4.12.6 Экспорт архива	98
4.13 Отправка реакций и событий в ПК Интеллект по HTTP-запросу	100
5. Заключение	100
6. ПРИЛОЖЕНИЕ 1. Описание структуры ddi-файла	101
7. ПРИЛОЖЕНИЕ 2. Объявление классов NissObjectDLLExt и CoreInterface	102

# Руководство по интеграции ПК Интеллект. Введение.

В данном документе приведена информация, позволяющая обеспечить взаимодействие ПК *Интеллект* с внешними системами. ПК *Интеллект* предоставляет следующие интерфейсы для решения данной задачи:

1. Интерфейс IIDK. Используется для внедрения в систему функциональных модулей, обеспечивающих решение следующих задач:
  - а. Добавление нового охранного оборудования в систему.
  - б. Реализация новых сервисных функций (управление охранным оборудованием).Этапы интеграции модулей рассмотрены на примере демонстрационного модуля *DEMO*, исходные файлы которого приложены к документации. Скачать модуль *DEMO* можно на странице [Руководство по интеграции ПК Интеллект](#).
2. Элемент управления ActiveX *CamMonitor.ocx*. Данный компонент является полным аналогом интерфейсного объекта **Монитор видеонаблюдения**. Он позволяет управлять камерами, просматривать архив и т.д.
3. HTTP API. Данный программный интерфейс позволяет отправлять команды и получать данные от ПК *Интеллект* при помощи HTTP-запросов.

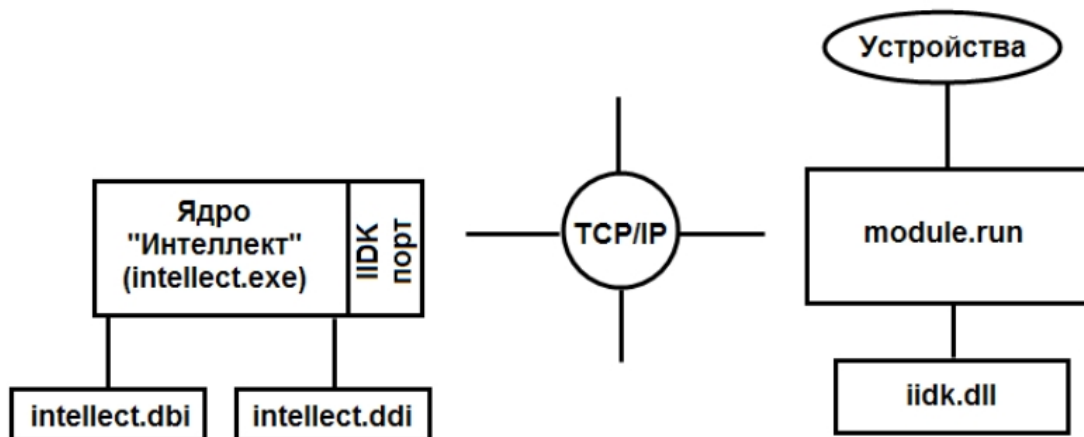
## Интеграция аппаратно-программных модулей

### INTELLECT INTEGRATION DEVELOPER KIT (IIDK)

#### Общие сведения об IIDK

#### Назначение IIDK

Возможность расширять систему заложена в архитектуру программного комплекса *Интеллект*, предусматривающую межзадачное взаимодействие ядра системы с функциональными модулями (смежными информационными системами) через коммуникационную среду TCP/IP. Схема взаимодействия ядра ПК *Интеллект* с внешним программным обеспечением (функциональным модулем) приведена на рисунке.



Взаимодействие ядра системы с внешним программным обеспечением выполняется посредством обмена сообщениями в коммуникационной среде, реализованного с помощью *IIDK*.

*Intellect Integration Developer Kit (IIDK)* представляет собой комплект средств разработки, используемый для интеграции охранного оборудования сторонних производителей с ПК *Интеллект*. Данный инструмент позволяет быстро и эффективно расширять систему, добавляя функциональные модули, поддерживающие новое оборудование или новые

сервисные функции.

## Требования к разработчику

Для использования IIDK требуется:

1. знание языка программирования C/C++ ;
2. знание основ программирования в Win32;
3. наличие среды разработки (*Microsoft Visual C++*, *C++ Builder*, *DELPHI* и др.), поддерживающей работу с dll-файлами.

### **Примечание.**

Создавая lib-файл в C++ Builder 5 при помощи утилиты implib.exe, необходимо указать ключ '-a'.

## Состав IIDK

IIDK включает в себя следующие средства разработки:

1. *iidk.ocx* – элемент управления ActiveX. Данный файл при установке ПК *Интеллект* помещается в папку Windows\System32 и регистрируется в операционной системе.
2. *ddi.exe* – программа для просмотра и редактирования DDI- и DBI- файлов. Располагается в папке <Директория установки ПК *Интеллект*>\Tools.

## Подключение к ПК Интеллект

### Параметры подключения

Взаимодействие ядра ПК *Интеллект* с функциональными модулями (смежными информационными системами) осуществляется со следующими параметрами подключения:

1. Номер порта.
  - а. Для видеоподсистемы – порт 900.
  - б. Для объекта **Интерфейс IIDK** – порт 1030.
  - с. Для объекта **Банкомат** – порт 1009.

### **Примечание.**

Для подключения банкоматов можно также использовать не порт 1009 (АТМ), а порт 1030 (IIDK), в таком случае объект **Банкомат** будет отображаться в дереве оборудования со значком красного креста. При этом в дереве оборудования должен быть создан объект **Интерфейс IIDK**.

2. IP-адрес компьютера, на котором функционирует ядро ПК *Интеллект*.
3. ID – идентификатор объекта подключения.

### **Внимание!**

Для подключения к видеоподсистеме (порт 900) id должен быть больше 1 и не должен совпадать с id созданных в системе объектов **Интерфейс IIDK**. Для подключения к объекту **Интерфейс IIDK** (порт 1030) id равен идентификационному номеру объекта, заданному в диалоговом окне настройки ПК *Интеллект*.

### **Примечание.**

Если требуется подключиться к серверу (объекту **Интерфейс IIDK**) с удаленного компьютера, не обязательно устанавливать ПК *Интеллект* на удаленный компьютер, но необходимо добавить этот компьютер в конфигурацию ПК *Интеллект* на сервере (на вкладке **Оборудование** диалогового окна **Настройка системы**), и именно на базе созданного объекта **Компьютер** создать объект **Интерфейс IIDK**. В таком случае в параметре IP функции Connect следует указывать адрес сервера, а в параметре ID идентификатор указанного объекта Интерфейс IIDK. Следует учитывать, что объект **Компьютер**, соответствующий удаленному компьютеру, будет помечен в дереве объектов красным крестом.

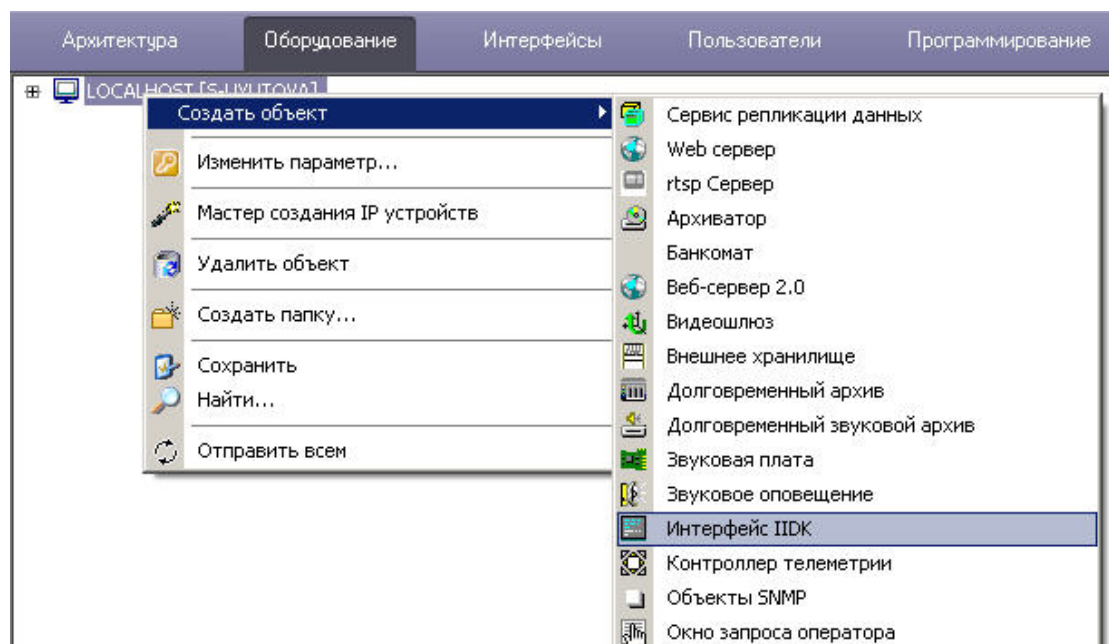
**Примечание.**  
Если создан mdl-файл (см. раздел *Разработка MDL-файла*), для подключения к ядру ПК *Интеллект* объект **Интерфейс IIDK** в системе не создается. В качестве идентификатора подключения передается пустая строка, то есть id равен "".

## Объект Интерфейс IIDK

Объект **Интерфейс IIDK** позволяет управлять всеми элементами системы. Объект **Интерфейс IIDK** создается на базе объекта **Компьютер** в дереве объектов ПК *Интеллект*.

**Примечание**  
Для использования объекта **Интерфейс IIDK** данная функциональная возможность должна быть разрешена в ключе активации.

**Примечание**  
Если ПК *Интеллект* запущен в демонстрационном режиме, объект **Интерфейс IIDK** будет активирован после подключения функционального модуля к ядру системы (см. раздел *Connect*).



В случае использования объекта **Интерфейс IIDK** панели настройки для интегрируемых функциональных модулей (смежного программного обеспечения) не создаются.

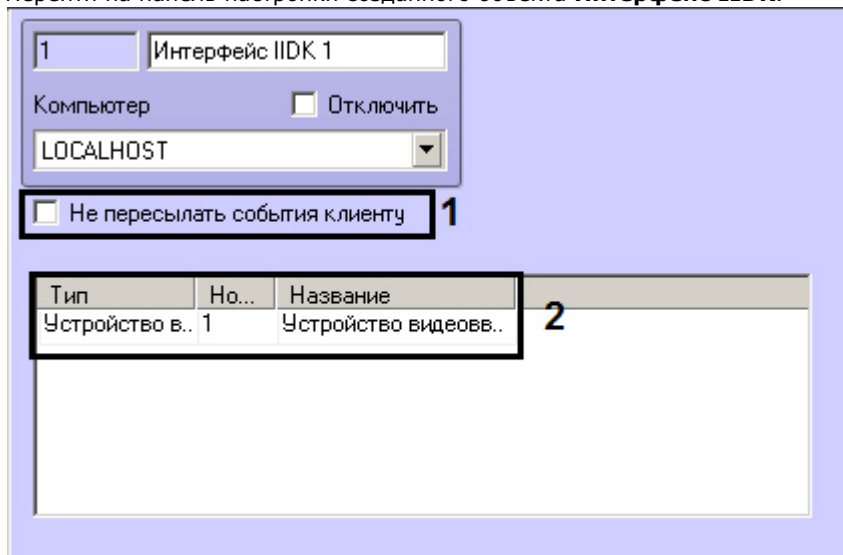
При использовании распределенной архитектуры ПК *Интеллект* объект **Интерфейс IIDK** должен быть создан на компьютере, содержащем программное ядро, к которому выполняется подключение. В случае, если подключение выполняется к компьютеру, на котором установлено *Рабочее место мониторинга*, то в параметрах подключения требуется указывать ip-адрес *Сервера* или *Рабочего места администратора*.

## Настройка пересылки событий через объект Интерфейс IIDK

Объект **Интерфейс IIDK** позволяет настроить фильтрацию событий, передаваемых подключаемым клиентским приложениям.

Настройка фильтрации осуществляется следующим образом:

1. Перейти на панель настройки созданного объекта **Интерфейс IIDK**.

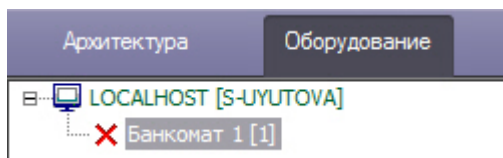


2. В случае, если не требуется передавать никакие сообщения клиентским приложениям, от которых не поступают запросы к ядру, установить флажок **Не пересылать события клиенту** (1).
3. В таблице (2) задать список объектов, события от которых следует передавать подключающимся клиентским приложениям.

Настройка фильтрации событий завершена.

## Особенности интеграции банкоматов. Объект Банкомат

Для передачи событий от ПО банкомата в ядро ПК *Интеллект* может быть использован объект **Банкомат**. Данный объект создается на базе объекта **Компьютер** на вкладке **Оборудование** диалогового окна **Настройка системы** вместо объекта **Интерфейс IIDK**.



Объект **Банкомат** позволяет отображать события банкомата ("Вставлена карта", "Отдана карта" и др.) в протоколе событий. Данные события также могут быть использованы для наложения титров на видео, настройки реакций и т.д.

**Примечание.**  
Для событий, связанных с картой клиента, имеется возможность передавать маскированный номер банковской карты в параметре `param0`.



Список доступных для использования событий объекта **Банкомат** можно узнать при помощи утилиты ddi.exe. Работа с данной утилитой описана в документе [Руководство Администратора](#), раздел [Утилита редактирования шаблонов баз данных](#) и файла внешних настроек ddi.exe.

Способ подключения и синтаксис сообщений для объекта **Банкомат** такие же, как для объекта **Интерфейс IIDK**, однако для отправки сообщений используется порт 1009 (см. также [Параметры подключения](#) и [Синтаксис сообщений \(900 порт\)](#)).

## Функции IIDK

### Connect

Для взаимодействия функционального модуля с ПК *Интеллект* необходимо выполнить подключение к ядру системы с помощью следующей функции:

```
BOOL Connect (LPCTSTR ip, LPCTSTR port, LPCTSTR id, void (_stdcall *func)(LPCTSTR msg))
```

Описание параметров функции Connect приведено в таблице.

Параметр	Описание	Пример
LPCTSTR ip	ip- адрес компьютера с ядром системы	<pre>CString port = "900";  CString ip = "127.0.0.1";  CString id = "2";  BOOL IsConnect = Connect(ip, port, id, myfunc);  if (!IsConnect) {      // не удалось подключиться      AfxMessageBox("Error");  }</pre>
LPCTSTR port	порт TCP/IP, через которое происходит подключение	
LPCTSTR id	идентификатор подключения, для видео	
_stdcall *func)(LPCTSTR msg))	Callback-функция, принимающая сообщения от ПК <i>Интеллект</i>	

Функция возвращает TRUE, если подключение выполнено, иначе - FALSE.

Все сообщения, приходящие от ядра системы, принимает Callback-функция.

Пример объявления Callback-функции:

```
void _stdcall myfunc(LPCTSTR str)
{
    printf("\r\nReceived:%s\r\n\r\n",str);
}
```

**Примечание.** `Void _stdcall myfunc` вызывается в отдельном потоке (не в контексте основного потока приложения).

Разбор получаемых сообщений устанавливается разработчиком в соответствии с требованиями интеграции.

## SendMsg

Для передачи сообщения ядру системы используется функция:

```
BOOL SendMsg (LPCTSTR id, LPCTSTR msg)
```

Описание параметров функции SendMsg приведено в таблице.

Параметр	Описание	Пример
LPCTSTR id	идентификатор подключения, указанный при вызове функции <b>Connect</b>	<pre>CString port = "900"; CString ip = "127.0.0.1"; CString id = "2"; BOOL IsConnect = Connect(ip, port, id, myfunc); if (!IsConnect) {     // не удалось подключиться</pre>

LPCTSTR msg	текст сообщения	<pre>AfxMessageBox("Error");  Return;  }  SendMsg(id,"CAM 1 REC"); // поставить камеру 1 на запись  Disconnect (id);</pre>
-------------	-----------------	--

Если сообщение отправлено, функция возвращает TRUE, иначе – FALSE.

## Disconnect

Каждое созданное соединение должно быть разорвано с помощью функции **Disconnect**:

```
void Disconnect (LPCTSTR id)
```

где **LPCTSTR id** – идентификатор подключения, указанный при вызове функции **Connect**.

Если разрыв соединения осуществляется со стороны ПК *Интеллект*, то Callback-функция принимает значение **DISCONNECTED**.



### Примечание.

Пример использования функции **Disconnect** приведен в разделе [SendMsg](#).

## Другие функции

### На странице:

- [Connect3](#)
- [SendReactToCore](#)
- [IsConnected](#)
- [Connect4](#)
- [SendData4](#)
- [SendFile](#)
- [GetMsg](#)

Ниже перечислены дополнительные функции, объявленные в заголовочном файле iidk.h. Из них не рекомендуются к использованию функции [Connect4](#), [SendData4](#), [SendFile](#), [GetMsg](#). Они созданы для внутреннего пользования. Функция [Connect2](#) не используется.

### **Connect3**

```
BOOL Connect3(LPCTSTR ip, LPCTSTR port, LPCTSTR id, iidk_callback_func* lpfunc,  
DWORD user_param,int async_connect,DWORD connect_attempts)
```

<b>Параметр</b>	<b>Описание</b>
ip	IP-адрес Сервера ПК <i>Интеллект</i> ,
port	Порт TCP/IP, через которое происходит подключение
id	Идентификатор подключения slave, для видео
lpfunc	Callback-функция, принимающая сообщения от ПК <i>Интеллект</i>
user_param	Дополнительный параметр, который будет приходить в Callback-функцию, чтобы разделить слейвы, если функция одна на всех.
async_connect	0 - синхронный режим подключения, функция возвращает TRUE, если подключение выполнено -1 - асинхронный режим подключения, функция всегда возвращает FALSE, если подключение выполнено, то генерируется событие CONNECTED Любое другое значение - сначала используется синхронный режим, в случае неудачи асинхронный.
connect_attempts	Количество попыток подключения

### **SendReactToCore**

Функция предназначена для отправки реакции в указанное ядро.

```
BOOL SendReactToCore(LPCTSTR id, LPCTSTR msg)
```

<b>Параметр</b>	<b>Описание</b>
id	Идентификатор подключения ядра
msg	Отправляемое сообщение. Формат сообщения аналогичен <a href="#">SendMsg</a> .

### **IsConnected**

IsConnected возвращает TRUE, если в данный момент клиент подключен к серверу.

```
BOOL IsConnected();
```

#### **Connect4**

```
BOOL Connect4(LPCTSTR ip, LPCTSTR port, LPCTSTR id, iidk_callback_func* lpfunc,  
iidk_frame_callback_func* lpframe_func, iidk_user_data_func* iidk_user_data_func,  
DWORD user_param,int async_connect,DWORD connect_attempts);
```

Параметр	Описание
ip	IP-адрес Сервера ПК <i>Интеллект</i> ,
port	Порт TCP/IP, через которое происходит подключение
id	Идентификатор подключения ядра, для видео
lpfunc	Callback-функция, принимающая сообщения от ПК <i>Интеллект</i>
lpframe_func	Callback-функция, принимающая видеокадры
iidk_user_data_func	Callback-функция для данных, посылаемых при помощи функции SendData4
user_param	Дополнительный параметр, который будет приходить в Callback-функцию, чтобы разделить ядра, если Callback-функция одна на все ядра.
async_connect	0 - синхронный режим подключения, функция возвращает TRUE, если подключение выполнено -1 - асинхронный режим подключения, функция всегда возвращает FALSE. Если подключение выполнено, то генерируется событие CONNECTED Любое другое значение - сначала используется синхронный режим, в случае неудачи асинхронный режим.
connect_attempts	Количество попыток подключения

#### **SendData4**

Данная функция используется для отправки CUserNetObject, ее назначение - отправка "сырых данных".

```
BOOL SendData4(LPCTSTR id, int nIdent,BYTE *pBuffer,DWORD dwSize);
```

Параметр	Описание
id	Идентификатор подключения ядра
nIdent	Уникальный идентификатор данных
pBuffer	Пересылаемые данные
dwSize	Размер массива данных

### **SendFile**

Функция служит для пересылки файла.

```
BOOL SendFile(LPCTSTR id, LPCTSTR file_from, LPCTSTR file_to)
```

Параметр	Описание
id	Идентификатор подключения ядра
file_from	Адрес, по которому находится файл для пересылки
file_to	Адрес, по которому следует записать файл.

### **GetMsg**

Функция служит для выборки пришедших сообщений, которые помещаются в очередь, если Callback-функция не указана.

```
BOOL GetMsg(LPTSTR msg, DWORD& cb)
```

Параметр	Описание
msg	Получаемое сообщение
cb	Длина сообщения

# Синтаксис отправляемых сообщений

## Синтаксис сообщений

Сообщения, отправляемые ядру, должны иметь следующий вид:

**CORE|DO\_REACT|source\_type<ТИП ОБЪЕКТА>,source\_id<ИДЕНТИФИКАТОР ОБЪЕКТА>,action<ДЕЙСТВИЕ> [ ,params<КОЛ-ВО ПАРАМЕТРОВ>,param0\_name<ИМЯ ПАРАМЕТРА\_0>,param0\_val<ЗНАЧЕНИЕ ПАРАМЕТРА\_0>]**

Ниже приведен синтаксис сообщения, содержащего 2 параметра.

**CORE|DO\_REACT|source\_type<ТИП ОБЪЕКТА>,source\_id<ИДЕНТИФИКАТОР ОБЪЕКТА>,action<ДЕЙСТВИЕ>,params<2>,param0\_name<ИМЯ ПАРАМЕТРА\_0>,param0\_val<ЗНАЧЕНИЕ ПАРАМЕТРА\_0>,param1\_name<ИМЯ ПАРАМЕТРА\_1>,param1\_val<ЗНАЧЕНИЕ ПАРАМЕТРА\_1>**

Описание параметров сообщения приведено в таблице.

Параметр	Описание
source_type<obj>	тип объекта (см. DDI-файл, секцию [OBJTYPE])
source_id<id>	идентификационный номер объекта, заданный при создании объекта в ПК <i>Интеллект</i> (см. дерево настроек в ПК <i>Интеллект</i> )
action<react>	действие (см. DDI-файл, секцию [REACT])
params<number>	число передаваемых параметров в десятичном формате
param0_name<str1>	имя параметра
param0_val<str2>	значение параметра

**Примечание.** Для работы с DDI-файлами предпочтительно использовать программу ddi.exe (см. раздел [Использование утилиты ddi.exe для работы с DDI-файлами](#)).

Пример. Отправление сообщения с командой перевода телеметрии в предустановку 4.

```
CString msg=  
  
"CORE|DO_REACT|source_type<TELEMETRY>,source_id<1.1>,action<GO_PRESET>,params<2>,param0_name<preset>,param0_val<4>,param1_name<tel_prior>,param1_val<2>";  
  
SendMsg(id,msg);
```

## Синтаксис сообщений (900 порт)

Сообщения, отправленные на 900 порт, передаются видеоподсистеме напрямую, поэтому сообщения имеют другой синтаксис.

Сообщения, отправляемые видеоподсистеме, имеют следующий вид:

**ТИП ОБЪЕКТА|ИДЕНТИФИКАТОР ОБЪЕКТА|ДЕЙСТВИЕ [|ПАРАМЕТР<ЗНАЧЕНИЕ>]**

Ниже описан синтаксис сообщения для видеоподсистемы, содержащего n-ое количество параметров.

**ТИП ОБЪЕКТА|ИДЕНТИФИКАТОР ОБЪЕКТА|ДЕЙСТВИЕ [|ПАРАМЕТР 1<ЗНАЧЕНИЕ>,ПАРАМЕТР 2<ЗНАЧЕНИЕ>,...,ПАРАМЕТР N<ЗНАЧЕНИЕ>]**



**Внимание!**

Через 900 порт можно управлять только объектами типа GRABBER, CAM и MONITOR.

Описание параметров сообщения представлено в таблице:

Параметр	Описание
Тип объекта	Тип объекта (GRABBER, CAM или MONITOR)
Идентификатор объекта	Идентификационный номер объекта, заданный при создании объекта в ПК <i>Интеллект</i>
Действие	Действие (команда)
Параметр <Значение>	Имя параметра. В треугольных скобках задается значение параметра

Пример 1. Постановка камеры 1 на запись.

```
CString msg = "CAM|1|REC";  
SendMsg (id,msg);
```

Пример 2. Запись видео со всех видеокамер на локальный диск «C:».

```
CString msg = "GRABBER|1|SET_DRIVES|drives<C:\>" ;  
SendMsg(id,msg);
```



**Примечание**

Для выполнения команды **SET\_DRIVES** необходимо указать идентификационный номер любой устройства видеоввода, созданной в системе.



**Примечание**




Команда **SET\_DRIVES** не меняет настройки записи видеосигнала в архив, заданные в системе.

## Использование классов **Event** и **React**

Для работы с сообщениями можно использовать прилагаемые классы: *Event* и *React*, объявленные в файле `msg.h`.

Сообщение, составленное без использования классов	Сообщение, составленное с помощью класса <code>React</code>
<pre>CString msg = "CORE  DO_REACT source_type&lt;TELEMETRY&gt;,source_id&lt;1.1&gt;, action&lt;GO_PRESET&gt;,params&lt;2&gt;,param0_name&lt;preset&gt;,param0_val&lt;4&gt;, param1_name&lt;tel_prior&gt;,param1_val&lt;2&gt;"; SendMsg(id,msg);</pre>	<pre>React react("TELEMETRY","1.1","GO_PRESET"); react.SetParamInt("preset",4); react.SetParamInt("tel_prior",2); SendMsg(id,react.MsgToString().c_str());</pre>

 **Примечание.**  
Файлы `msg.h` и `msg.cpp` содержатся в папке `Misc`.

## Примеры управления объектами системы

 **Внимание!**  
Команды и параметры объектов подробно описаны в документе [Руководство по программированию](#).

## Добавление, изменение и удаление объектов системы

### На странице:

- [Добавление пользователя в отдел](#)
- [Добавление и удаление устройства видеоввода](#)

Добавление, изменение и удаление объектов системы выполняется с помощью команд:

1. **CORE||CREATE\_OBJECT** – для создания нового объекта.
2. **CORE||UPDATE\_OBJECT** – для изменения существующего объекта или создания нового.
3. **CORE||DELETE\_OBJECT** – для удаления объекта.

### Добавление пользователя в отдел

Ниже приведено сообщение, в результате обработки которого в отдел будет добавлен пользователь с заданными параметрами:

```
CORE||CREATE_OBJECT|objtype<PERSON>,objid<12>,parent_id<1>,name<Иванов Иван Иванович>,core_global<0>,params<11>,param0_name<facility_code>,param0_val<122>,param1_name<card>,param1_val<1234>,param2_name<pin>,param2_val<>,param3_name<comment>,param3_val<Начальник отдела кадров>,param4_name<is_locked>,param4_val<0>,param5_name<is_apb>,param5_val<0>,param6_name<level_id>,param6_val<*>,param7_name<person>,param7_val<>,param8_name<_creator>,param8_val<1>,param9_name<expired>,param9_val<>,param10_name<temp_card>,param10_val<>
```

## Добавление и удаление устройства видеоввода

Добавление объекта выполняется с помощью команды **UPDATE\_OBJECT**, если в системе отсутствует объект с указанными значениями для параметров **objtype** и **objid**.

```
CORE||UPDATE_OBJECT|objtype<GRABBER>,objid<12>,core_global<0>,parent_id<SLAVAXP>,name<Устройство видеоввода 1>,params<5>,param0_name<format>,param0_val<NTSC>,param1_name<mode>,param1_val<1>,param2_name<chan>,param2_val<2>,param3_name<type>,param3_val<FX 4>,param4_name<resolution>,param4_val<0>
```

Получив следующее сообщение, система изменит имя созданного объекта:

```
CORE||UPDATE_OBJECT|objtype<GRABBER>,objid<12>,core_global<0>,parent_id<SLAVAXP>,name<Устройство 2>
```

Для удаления объекта и всех его дочерних объектов используется команда **DELETE\_OBJECT**:

```
CORE||DELETE_OBJECT|objtype<GRABBER>,objid<12>
```

## Особенности работы с системой в многопользовательском режиме

На удаленном компьютере должен быть установлен (тип установки – **Клиент**) и запущен ПК *Интеллект*, для того чтобы обмениваться сообщениями с Сервером.

Если в ПК *Интеллект* созданы пользователи и настроены права доступа, то передаваемые сообщения, требующие ответа от ядра системы, должны содержать параметр **receive\_r\_id<ID>**, где ID – это идентификационный номер объекта **Интерфейс IIDK** в системе.

```
CORE||GET_CONFIG|objtype<CAM>,objid<1>,receiver_id<1>
```

// Возвращает параметры объекта «Камера 1»

## Определение компьютера, на котором был выгружен ПК Интеллект (через 1030 порт)

В случае выгрузки ПК *Интеллект* в Callback-функцию придет сообщение, где параметру **action** присвоено значение **DISCONNECTED**:

```
ACTIVEX|12|EVENT|SOCKET<>,MMF<>,objaction<DISCONNECTED>,TRANSPORT_TYPE<MMF>,core_global<1>,action<DISCONNECTED>,module<slave.exe>,objtype<SLAVE>,_slave_id<SLAVAXP.12>,objid<SLAVAXP>,owner<SLAVAXP>,TRANSPORT_ID<1111>,time<12:41:16>,date<23-09-02>
```

Данное сообщение содержит имя компьютера, на котором был выгружен ПК *Интеллект*, дату и время, когда это действие произошло.

## Вывод видеокамеры на монитор

Система удалит все камеры с монитора и вызовет указанную видеокамеру, получив следующее сообщение:

```
CORE||DO_REACT|source_type<MONITOR>,source_id<1>,action<REPLACE>,params<4>,param0_name<slave_id>,param0_val<SLAVA>,param1_name<cam>,param1_val<1>,param2_name<control>,param2_val<1>,param3_name<name>,param3_val<>
```

При подключении через 900 порт действие, описанное выше, выполняется с помощью сообщения:

```
MONITOR|1|REPLACE|slave_id<SLAVA>,cam<1>,control<1>
```

## Получение параметров объекта (через 1030 порт). GET\_CONFIG

Пример использования команды **GET\_CONFIG** приведен ниже.

```
CORE||GET_CONFIG|objtype<CAM>,objid<1>
```

В возвращаемом сообщении будут содержаться все параметры указанного объекта:

```
ACTIVEX|12|OBJECT_CONFIG|rec_priority<0>,mask0<>,decoder<0>,mask1<>,flags<>,mask2<>,compression<3>,sat_u<5>,mask3<>,proc_time<>,hot_rec_period<>,mask4<>,telemetry_id<>,manual<1>,region_id<1.1>,contrast<5>,md_mode<0>,md_size<5>,audio_type<>,pre_rec_time<0>,config_id<>,bright<7>,alarm_rec<0>,audio_id<>,rec_time<>,hot_rec_time<2>,activity<>,mux<0>,parent_id<1>,objtype<CAM>,type<>,__slave_id<SLAVAXP.12>,objid<1>,name<Камера 1>,objname<Камера 1>,color<1>,priority<0>,md_contrast<5>
```

**Примечание.**

Если убрать параметр **objid**, то в Callback-функцию вернется конфигурация всех объектов заданного типа.

Пример. Получить данные пользователя по его идентификатору.

```
CORE||GET_CONFIG|objtype<PERSON>,objid<1>
```

В ответ придет сообщение, среди параметров которого находится требуемая информация, такая как имя пользователя, номер карты доступа и прочее:

```
ACTIVEX|1|OBJECT_CONFIG|pnet3_sound<0>,galaxy_dual_focus<0>,auto_pass_type<>,galaxy_pin_change<0>,external_id<>,card_date<26.05.2017 10:57:06>,galaxy_tag_link<0>,rubeg8_zone_id<>,levels_times<>,expired<>,hid_escort_id<>,objtype<PERSON>,level2_id<>,galaxy_group_choice<0>,who_level<>,hid_use_extended_access<0>,visit_purpose<>,card<1234>,email<>,galaxy_timer_schedule<0>,galaxy_menu_option<0>,aiu_holiday<0>,area_id<>,aiu_alarm<0>,objname<Пользователь 1>,surname<>,who_card<>,auto_brand<>,pnet3_alarm<0>,card_loss<0>,facility_code<432>,galaxy_temp_code<0>,post<>,when_area_id_changed<>,drivers_licence<>,bolid_in_device<0>,pnet3_acs_counter<0>,temp_levels_times<>,temp_card<>,pnet3_no_entry<0>,location<>,temp_level_id<>,patronymic<>,teleph_work<>,department<>,galaxy_keypad<0>,_TRANSPORT_ID<>,finished_at<>,aiu_ksd_type<>,all_param<>,galaxy_template<0>,tabnum<>,parent_id<1>,pur<>,galaxy_duress<0>,pnet3_no_exit<0>,galaxy_dual<0>,hid_pin_exempt<0>,pnet3_counter<0>,whence<>,schedule_id<>,hid_enable_pin_commands<0>,galaxy_dual_access<0>,pnet3_block<0>,passport<>,person<>,galaxy_menu_choice<0>,flags<0>,auto_number<>,phone<>,pin<>,rubeg8_AccessToBCPTimeZoneNumber<>,begin_temp_level<>,pnet3_master<0>,aiu_mark<0>,end_temp_level<>,visit_birthplace<>,galaxy_menu_level<>,visit_document<>,pnet3_guard<0>,pnet3_black<0>,aiu_kso_type<>,is_apb<0>,name<Пользователь 1>,pnet3_temp<0>,started_at<>,level_id<>,_marker<>,bolid_user_type<>,owner_person_id<>,card_type<>,is_active_temp_level<0>,begin<>,hid_line_tag<>,guid<{5B358685-E041-E711-BCC6-DA0AE28E0C17}>,pnet3_guest<0>,is_locked<0>,objid<1>,marketing_info<>,comment<>,aiu_vpu_arm<0>,visit_reg<>,bolid_in_pku<0>,pnet3_2cards_mask<0>
```

**Примечание.**

Данные пользователя также можно получить напрямую через запрос к базе данных ПК *Интеллект* из таблицы OBJ\_PERSON. В этом случае можно использовать для выбора требуемого пользователя также номер карты. Подробнее о работе с базой данных ПК *Интеллект* см. [Руководство Администратора](#), раздел [Приложение 4. Администрирование базы данных программного комплекса Интеллект](#).

## Получение информации о состоянии объекта. GET\_STATE и GET\_LIST

Для получения информации о состоянии объекта используется команда **GET\_STATE**:

```
CORE||GET_STATE|objtype<CAM>,objid<1>
```

В результате возвратится строка:

```
ACTIVEX|12|OBJECT_STATE|objtype<CAM>,__slave_id<SLAVAXP.12>,objid<1>,state<DISARM_DETACHED>
```

Состояние указанного объекта будет представлено значением параметра **state** – одно из состояний, указанных в DDI-файле для выбранного объекта.

При подключении через 900 порт запрос состояний объектов выполняется с использованием команды **GET\_LIST**:

```
CAM||GET_LIST
```

**Примечание.**

Независимо от того, указан идентификационный номер объекта или нет, команда возвратит состояния всех объектов заданного типа.

Возвращаемые сообщения имеют вид:

**CAM|1|SETUP|rec\_priority<0>,is\_armed<0>,is\_recorded<0>, bt<0>, slave\_id<SLAVAXP>, compression<3>,sat\_u<5>, proc\_time<0>, hot\_rec\_period<0>, manual<1>, telemetry\_id<>, is\_detached<1>, contrast<5>, md\_size<5>,md\_mode<0>, is\_alarmed<0>, audio\_type<>, pre\_rec\_time<0>, bright<7>, audio\_id<>, rec\_time<0>, alarm\_rec<0>, hot\_rec\_time<2>, mux<0>, parent\_id<1>, \_\_slave\_id<SLAVAXP>, priority<0>, mask<>, color<1>,md\_contrast<5>, is\_ring<1>**

Состояния в сообщении представлены следующим образом: **is\_state<val>**, где **state** – имя состояния объекта (см. DDI-файл); **val** – принимает значение 1, если объект находится в соответствующем состоянии, иначе – 0.



**Примечание.**

Параметр **is\_ring<>** говорит о том, ведет ли камера запись в архив по кольцу.

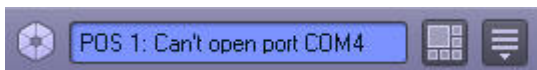
## Вывод информационного сообщения. SET\_STATE

Для вывода информационного сообщения на дисплей главной панели управления ПК *Интеллект* используется команда SET\_STATE:

**CORE||SET\_STATE|name<POS 1>,value<Can't open port COM4>**

Результат обработки сообщения системой представлен на рисунке.

Отображение сообщения на дисплее главной панели управления ПК *Интеллект*:



Удаление информационного сообщения с дисплея выполняется следующим образом:

**CORE||SET\_STATE|name<POS 1>,value<>**

## Работа с живым и архивным видео

Для получения живого видео с Камеры 1 следует отправить на порт 900 сообщение:

**CAM|1|START\_VIDEO|compress<1>**

Здесь **compress<>** – степень компрессии, от 0 до 5. В ответ на это сообщение начнут приходить кадры видео. Пример программной обработки поступающих кадров можно найти в демо-комплекте, доступном для скачивания на странице [Руководство по интеграции аппаратно-программных модулей](#).

Для получения архивного видео с Камеры 1 следует отправить на порт 900 следующие сообщения:

**CAM|1|ARCH\_FRAME\_TIME|time<dd-mm-yy HH:MM:SS.FFF>** – для установки времени, начиная с которого требуется просматривать архив.

**CAM|1|PLAY|compress<>** – для получения архивного видео. Работа с архивным видео осуществляется таким же образом, как с живым.

Для того, чтобы получить список временных интервалов, содержащих видеозаписи за определенную дату, необходимо послать на порт 900 следующее сообщение:

**CAM|id|ARCH\_GET\_INTERVALSREC|date<>**

Параметр **date<>** может принимать значение **date<dd-mm-yy>** или быть оставлен пустым. В первом случае будут запрошены интервалы за указанную дату, во втором – даты,

за которые присутствует архив. В результате будет получено сообщение вида

**Event: CAM|id|SET\_INTERVALSREC|intervals<>,date<>**

Значение параметра intervals<> имеет следующий вид: intervals<begin1 end1\nbegin2 end2...\nbeginN endN|date1\ndate2...\ndateN\n>

Время начала и время конца разделяется одним пробелом (код 0x20), интервалы отделяются друг от друга символом переноса строки "\n"(код 0x0A).

- begin1, begin2, ... beginN – времена начал интервалов в формате HH:MM:SS (возвращается, если запрошена точная дата).
- end1, end2, ... endN – времена концов интервалов в формате HH:MM:SS(возвращается, если запрошена точная дата).
- date1, date2, ... dateN – даты для которых присутствуют записи в архиве (возвращается, если поле date в запросе пусто или отсутствует).

Параметр date<dd-mm-yy> – это дата, за которую запрашивались интервалы, или пустое значение (date<>), если запрашивались даты за весь период.

## Управление телеметрией

Управление телеметрией через IIDK осуществляется при помощи обычных реакций, описанных в [Руководстве по программированию](#) в разделе **TELEMETRY**, например:

**CORE||DO\_REACT|source\_type<TELEMETRY>,source\_id<1.1>,action<LEFT>,params<1>,param0\_name<tel\_prior>,param0\_val<3>** – сообщений на порт 1030 для поворота объектива камеры влево с высоким приоритетом.

**TELEMETRY|1.1|LEFT|speed<2>,tel\_prior<3>** – реакция на порт 1030 для поворота объектива камеры влево с высоким приоритетом и средней скоростью.

## Операции со слоем карты

Команда задания размера и положения значка объекта **Камера 1** на слое 1 выполняется одним из следующих способов:

1. Посылкой сообщения на порт 1030 **CORE||DO\_REACT|source\_type<MAPPLAYER>,source\_id<1>,action<CUSTOMIZE\_OBJECT>,params<7>,param0\_name<x>,param0\_val<200>,param1\_name<y>,param1\_val<200>,param2\_name<objtype>,param2\_val<CAM>,param3\_name<objid>,param3\_val<1>,param4\_name<a>,param4\_val<90>,param5\_name<w>,param5\_val<70>,param6\_name<h>,param6\_val<80>**

Здесь x, y, w, h – координаты и размер значка объекта на карте.

a – угол наклона значка.

2. Посылкой реакции на порт 1030 **MAPPLAYER|1|CUSTOMIZE\_OBJECT|x<200>,y<200>,objtype<CAM>,objid<1>,a<90>,w<70>,h<80>**

Вывод слоя 1 в окне интерактивной карты осуществляется одним из следующих способов:

1. Посылкой сообщения на порт 1030: **CORE||DO\_REACT|source\_type<MAPPLAYER>,source\_id<1>,action<ACTIVATE>**
2. Посылкой реакции на порт 1030: **MAPPLAYER|1|ACTIVATE"**

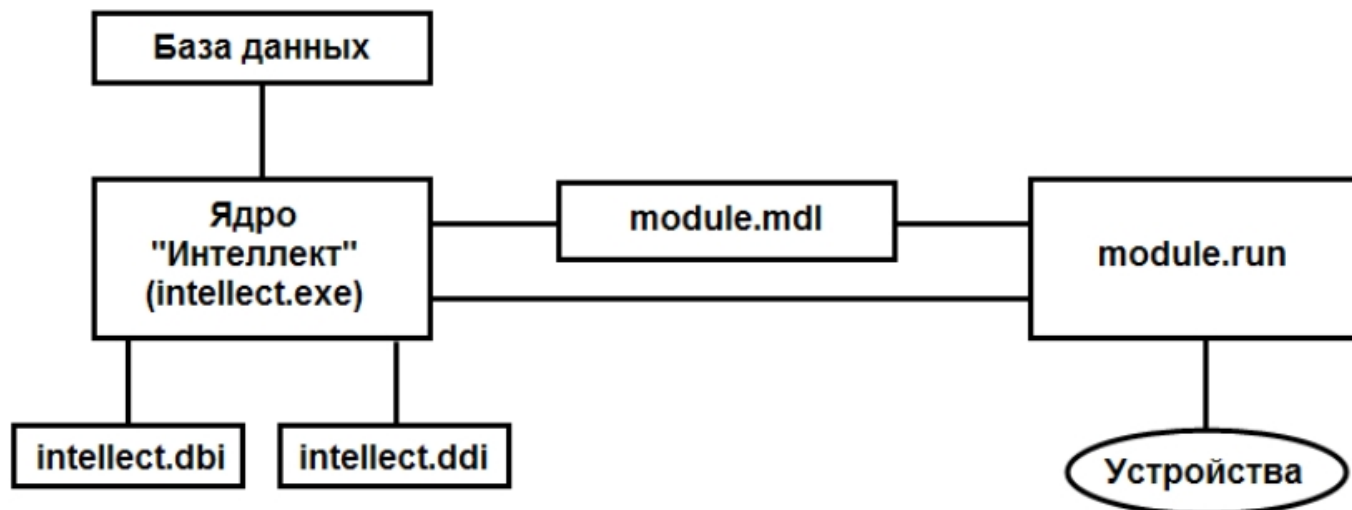
## Интеграция аппаратно-программных модулей с ПК Интеллект

### Общие сведения об интеграции аппаратно-программных модулей

Процесс интеграции аппаратно-программных (функциональных) модулей с ПК *Интеллект* состоит из следующих этапов:

1. Редактирование DBI-файла.
2. Редактирование DDI-файла.
3. Подготовка файла module.mdl, где module – имя интегрируемого модуля (данный файл является преобразованным DLL-файлом).
4. Подготовка исполнительного файла module.run, где module – имя интегрируемого модуля (этот файл является преобразованным exe-файлом).
5. Размещение module.mdl и module.run в каталоге *Интеллект\Modules*.

Схема взаимодействия функционального модуля с ядром системы представлена на рисунке.



DBI- и DDI-файлы содержат необходимую для функционирования ядра системы информацию об интегрированных функциональных модулях (объектах). DBI-файл содержит описание структуры конфигурационной базы данных ПК *Интеллект*. В DDI-файле хранится описание объектов и их параметров. При интеграции объекта в данные файлы заносят наименование, параметры интегрируемого объекта, связанные с ним системные события и реакции.

MDL-файлы обеспечивает работу с объектами одного типа: создание, изменение, удаление, изменение при настройке или в процессе работы параметров объекта и сохранение их в базе данных, выполнение некоторых специализированных операций с объектом. Также MDL-файл обеспечивает пересылку параметров созданных или измененных объектов исполнителю модулю (RUN-файлу) и хранит конфигурации настроечных панелей объектов.

Исполняемый RUN-файл осуществляет взаимодействие с устройствами, транслирует в ядро информацию о событиях, обеспечивает выполнение управления устройствами.

Далее описываются этапы интеграции модулей на примере демонстрационного модуля *DEMO*, эмулирующего работу с виртуальным оборудованием. Данный модуль включает в себя устройства с уникальными адресами для обращения к этим устройствам и их опроса. Таким образом, в системе будет существовать конфигурация, состоящая из 2 основных объектов: родительского объекта **DEMO** с параметром **COM-port** и дочернего объекта **DEMO\_DEVICE** с параметром **Address**. В системе возможно выполнение определенного набора действий с устройствами и передача всех происходящих в них событий ядру системы.

## Редактирование DBI-файла

Файл *intellect.dbi* содержит основной перечень таблиц и полей базы данных. Рекомендуется создавать собственный шаблон базы данных в отдельном файле – *intellect.xxx.dbi*, где *xxx* – уникальная часть имени файла. Использование отдельного файла позволяет избежать повторного включения таблиц и полей в случае обновления ПК *Интеллект*. При запуске программного комплекса DBI-файлы объединяются.

## Добавление объектов в intellect.dbi

Добавление объектов в *intellect.dbi* выполняется следующим образом:


1. Открыть в текстовом редакторе файл *intellect.dbi*, расположенный в корневом каталоге ПК *Интеллект*.
2. Добавить в *intellect.dbi* объекты. Для этого необходимо в квадратных скобках указать имя, используемое для идентификации объекта, и далее объявить его поля. Синтаксис объявления полей имеет вид:

<Имя поля>, <Тип> [, <Размер>]

 **Примечание.**  
**Размер** задается только полям с типом **CHAR**.

В таблице приведены обязательные поля для всех объектов ПК *Интеллект*.

Поле	Описание
id	Уникальный идентификатор объекта
name	Имя объекта
parent_id	Идентификатор родительского объекта
flags	Параметр для внутренних нужд системы

 **Внимание!**  
Поле **flags** не может использоваться внешними приложениями.

Допустимые типы данных описаны в таблице.

Тип данных	Описание
BIT	Используется для создания поля-флажка, принимающего логические значения «Да» или «Нет»
CHAR	Используется для полей, заполняемых небольшим количеством символов
DATETIME	Используется для полей, в которые вводятся дата и время. Маска для даты – гггг-мм-дд, для времени – чч:мм:сс.xxx
DOUBLE	Используется для полей, содержащих числа с плавающей запятой
INTEGER	Используется для полей, содержащих целые числа
TEXT	Используется для полей, содержащих текстовые строки

Для объектов демонстрационного модуля *DEMO*, кроме обязательных полей, добавлены поля:

- a. **port** – адрес COM-порта;
- b. **address** – адрес устройства.

Результат добавления объектов и объявления полей в *intellect.dbi* представлен на рисунке.

```
intellect - Блокнот
Файл  Правка  Формат  Вид  Справка
X, INTEGER
Y, INTEGER
W, INTEGER
H, INTEGER
exe, CHAR, 255
guid, UNIQUEIDENTIFIER

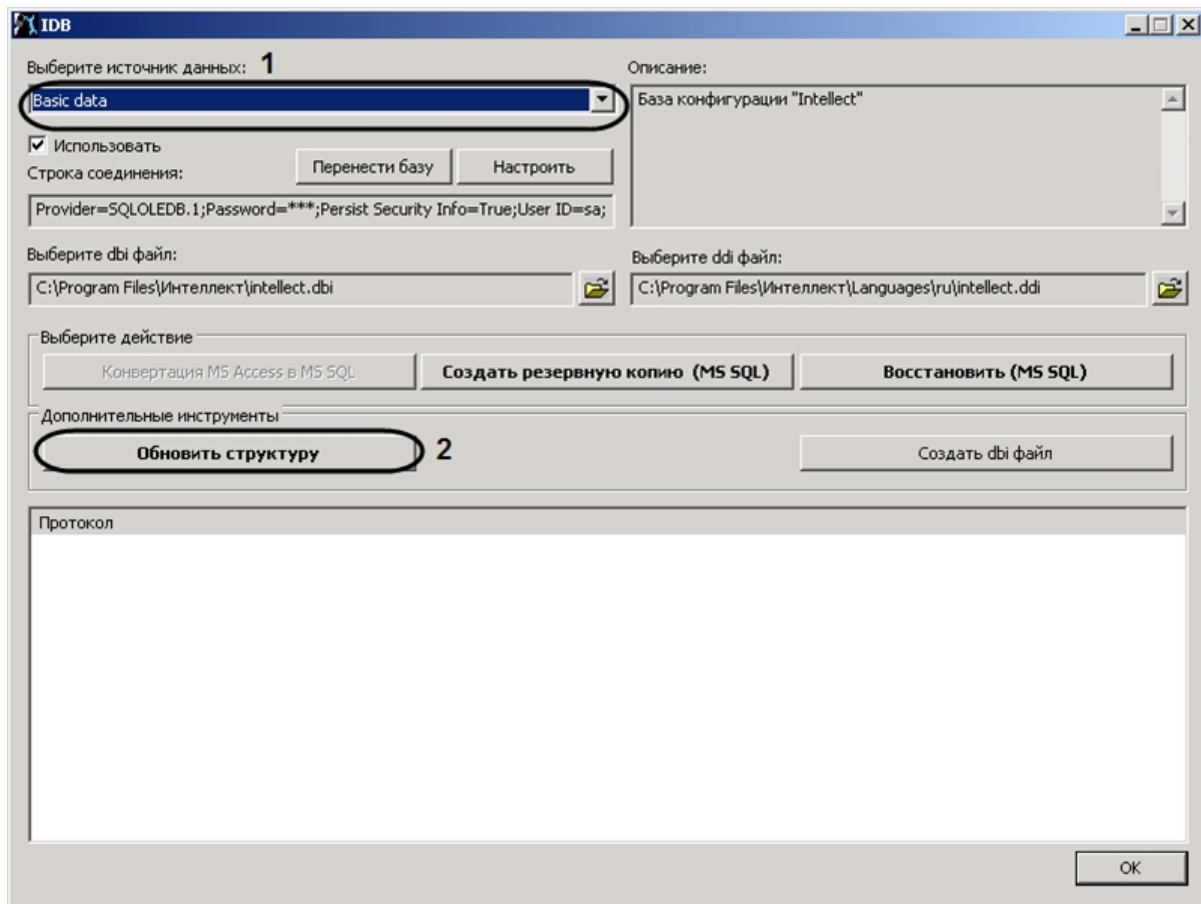
[OBJ_ZONE]
id, CHAR, 40
name, CHAR, 60
parent_id, CHAR, 40
flags, INTEGER
guid, UNIQUEIDENTIFIER

[OBJ_DEMO]
id, CHAR, 16
name, CHAR, 60
parent_id, CHAR, 16
port, CHAR, 5
flags, INTEGER

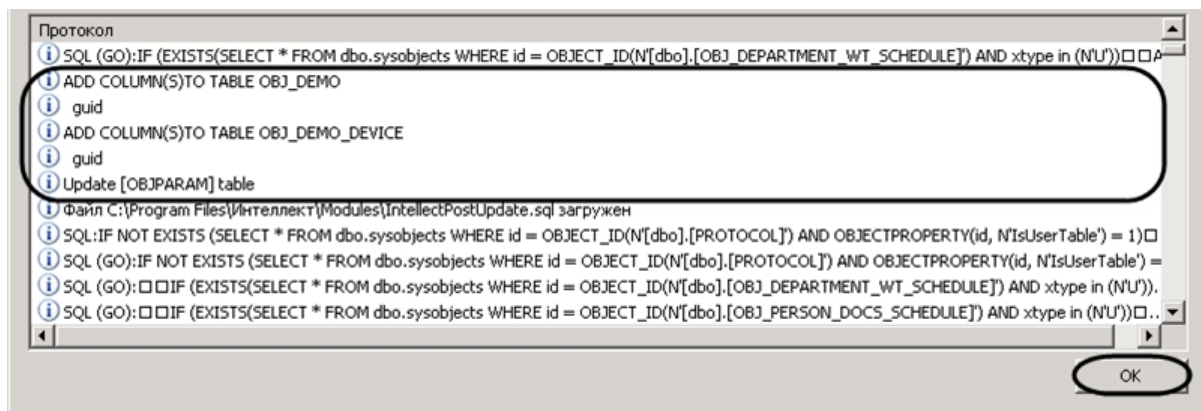
[OBJ_DEMO_DEVICE]
id, CHAR, 16
name, CHAR, 60
parent_id, CHAR, 16
address, INTEGER
flags, INTEGER
```

3. Сохранить изменения в файле intellect.dbi.
4. Запустить утилиту *idb.exe*, расположенную в корневом каталоге ПК *Интеллект*.





5. Из списка **Выберите источник данных:** выбрать **Basic data** (1).
6. Нажать кнопку **Обновить структуру** (2).  
Будет запущен процесс обновления структуры базы данных. Выполнение процесса отображается в окне **Протокол** утилиты *idb.exe*.



7. Нажать кнопку **OK** для завершения работы с утилитой *idb.exe*.

В результате обновления структуры будут созданы таблицы в базе конфигурации *Intellect*.

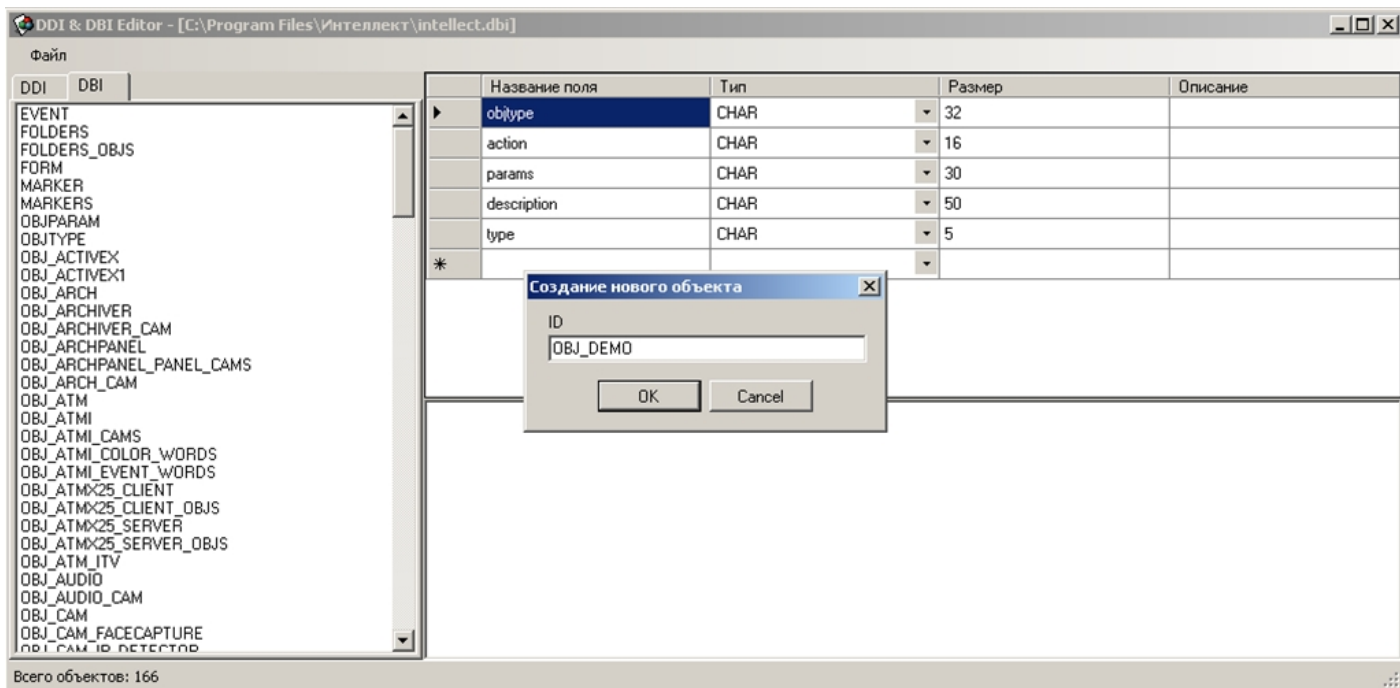
## Использование утилиты **ddi.exe** для работы с DBI-файлами

Для добавления объекта в DBI-файл с помощью утилиты *ddi.exe* необходимо выполнить следующие действия:

1. Запустить утилиту *ddi.exe*, расположенную в каталоге *Интеллект\Tools*.
2. В окне утилиты перейти на вкладку **DBI**.
3. В меню **Файл** выбрать пункт **Открыть**. В результате выполнения операции появится диалоговое окно **Открыть**.
4. Выбрать файл *intellect.dbi*, расположенный в корне директории установки ПК *Интеллект*. В утилите *ddi.exe* отобразится список объектов.
5. В контекстном меню списка объектов выбрать пункт **Добавить** для добавления нового объекта.

**Примечание.**  
Также возможно добавить объект с помощью клавиши **Insert**.

6. В открывшемся диалоговом окне ввести имя, используемое для идентификации объекта, в поле **ID** и нажать **OK**.



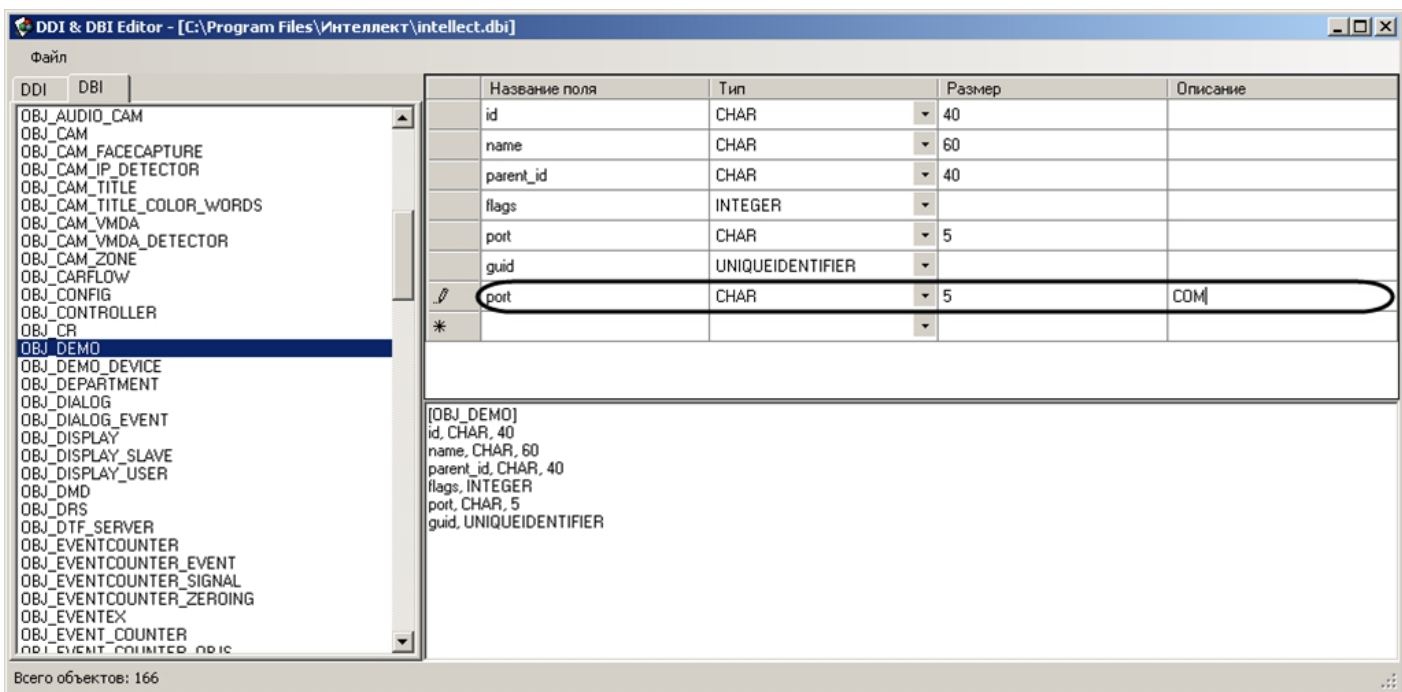
**Примечание.**

Созданному объекту будут автоматически добавлены обязательные поля (см. раздел [Добавление объектов в intellect.dbi](#)).

Добавление объекта в DBI-файл завершено.

Добавление поля выполняется следующим образом:

1. Выбрать созданный объект в левой части окна утилиты ddi.exe
2. Добавить строку с описанием нового поля в таблицу.



3. Сохранить внесенные изменения, выбрав в меню **Файл** пункт **Сохранить**.

Добавление поля выполнено.

**Внимание!** После изменения DBI-файлов требуется обновить структуру базы данных с помощью утилиты idb.exe (см. раздел [Добавление объектов в intellect.dbi](#)).

## Редактирование DDI-файла

DDI-файл представляет собой файл в формате xml, который содержит следующую информацию об объектах:

1. Реакции – действия, которые могут выполнять объекты.
2. События, которые могут генерировать объекты.
3. Состояния, в которых могут находиться объекты.
4. Правила перехода объектов из одного состояния в другое по определенным событиям.
5. Имена bmp-файлов, используемых для отображения объектов на *Карте*.

Файл intellect.ddi содержит свойства основных объектов ПК *Интеллект*. Для собственных объектов рекомендуется создавать отдельный файл – intellect.xxx.ddi, где xxx – уникальная часть имени файла. Использование отдельного файла позволяет избежать повторного включения свойств объектов в случае обновления ПК *Интеллект*. При запуске программного комплекса DDI-файлы объединяются.

В случае, если объект дублируется в нескольких ddi-файлах, при запуске ПК *Интеллект* применяются свойства объекта из последнего файла в соответствии с сортировкой файлов по имени. Например, если какой-либо объект описан в файлах intellect.xxx.ddi, intellect.xxx1.ddi и intellect.xxx2.ddi, будут применены свойства из файла intellect.xxx2.ddi.

## Добавление в intellect.ddi информации об объекте

В данном разделе приведен пример добавления в intellect.ddi информации об объекте **ДЕМО** с использованием текстового редактора.

Для добавления информации об объекте **ДЕМО** в intellect.ddi необходимо выполнить следующие действия:

1. Открыть в текстовом редакторе файл intellect.ddi, расположенный в каталоге *Интеллект\Languages\ru*.
2. В раздел **<DataSetDDI>** добавить дочерний элемент **<Objects>**, содержащий описание объекта.

```

<Objects>
  <ObjectName>DEMO</ObjectName>
  <VisibleName>Демо</VisibleName>
  <Events>
    <EventName>LOST</EventName>
    <EventDescription>Потеря связи</EventDescription>
    <IsSoundEnabled>>false</IsSoundEnabled>
    <IsNetworkDisabled>>false</IsNetworkDisabled>
    <IsProtocolDisabled>>false</IsProtocolDisabled>
    <IsWindowsLogEnabled>>false</IsWindowsLogEnabled>
  </Events>
  <Events>
    <EventName>RESTORE</EventName>
    <EventDescription>Восстановление связи</EventDescription>
    <IsSoundEnabled>>false</IsSoundEnabled>
    <IsNetworkDisabled>>false</IsNetworkDisabled>
    <IsProtocolDisabled>>false</IsProtocolDisabled>
    <IsWindowsLogEnabled>>false</IsWindowsLogEnabled>
  </Events>
  <Icons>
    <FileName>demo</FileName>
    <IconName>demo</IconName>
  </Icons>
  <States>
    <StateName>DETACHED</StateName>
    <ImgName>detached</ImgName>
    <IsStateFlashing>>false</IsStateFlashing>
  </States>
  <States>
    <StateName>NORMAL</StateName>
    <ImgName>normal</ImgName>
    <IsStateFlashing>>false</IsStateFlashing>
  </States>
  <Rules>
    <EventName>RESTORE</EventName>
    <FromStateName>DETACHED</FromStateName>
    <ToStateName>NORMAL</ToStateName>
  </Rules>
  <Rules>
    <EventName>LOST</EventName>
    <FromStateName>NORMAL</FromStateName>
    <ToStateName>DETACHED</ToStateName>
  </Rules>
</Objects>

```

 **Примечание**

Для объекта **DEMO** отсутствует раздел **<Reacts>**, так как данный объект не выполняет никаких действий.



#### Примечание

Элементы DDI-файла подробно описаны в разделе [ПРИЛОЖЕНИЕ 1. Описание структуры ddi-файла](#)

3. Сохранить изменения в файле `intellect.ddi`.

Внесение в `intellect.ddi` информации об объекте **DEMO** завершено.



#### Внимание!

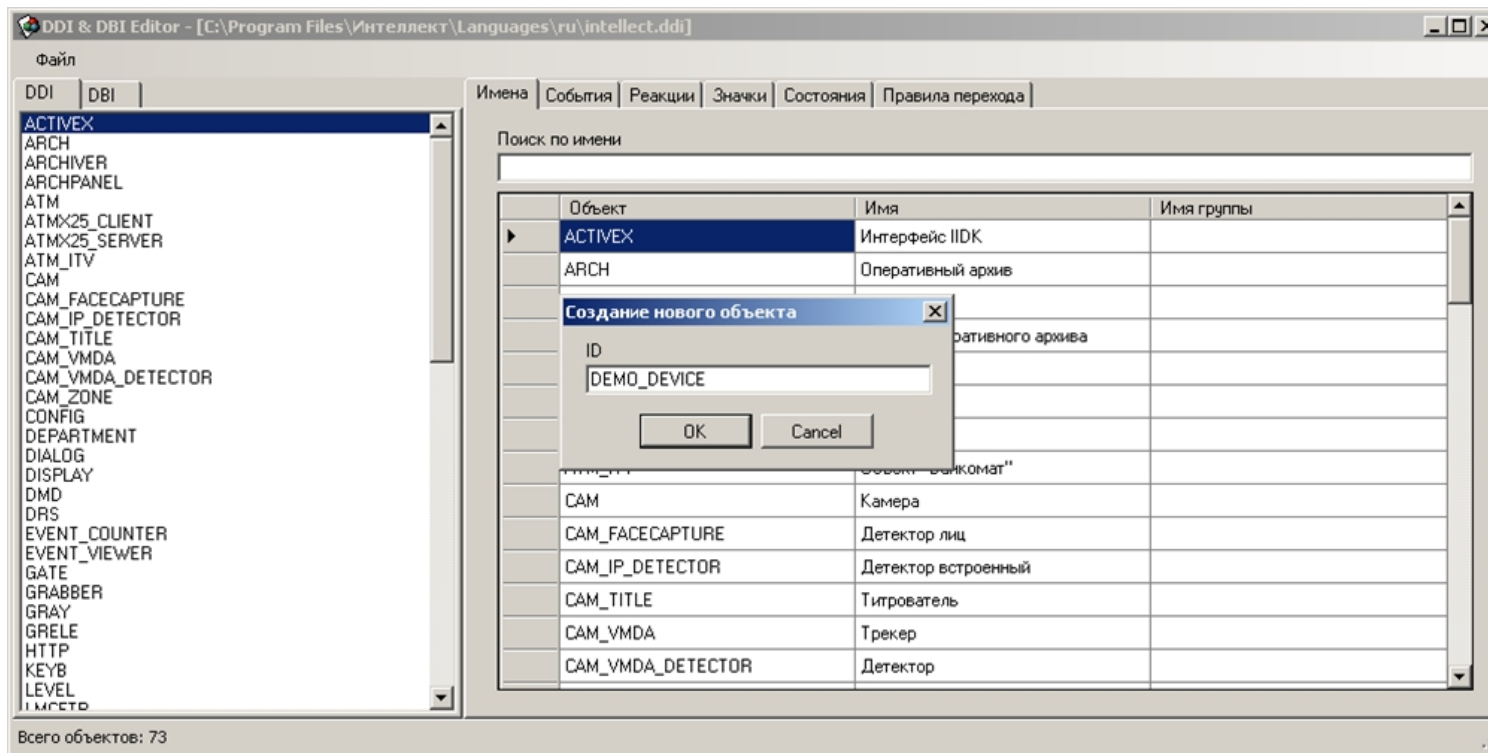
После изменения DDI-файлов требуется обновить структуру базы данных с помощью утилиты `idb.exe` (см. шаги 4-7 раздела [Добавление объектов в intellect.dbi](#)).

## Использование утилиты `ddi.exe` для работы с DDI-файлами

В данном разделе приведен пример добавления в `intellect.ddi` информации об объекте **DEMO\_DEVICE** с использованием утилиты `ddi.exe`.

Для добавления в `intellect.ddi` информации об объекте **DEMO\_DEVICE** необходимо выполнить следующие действия:

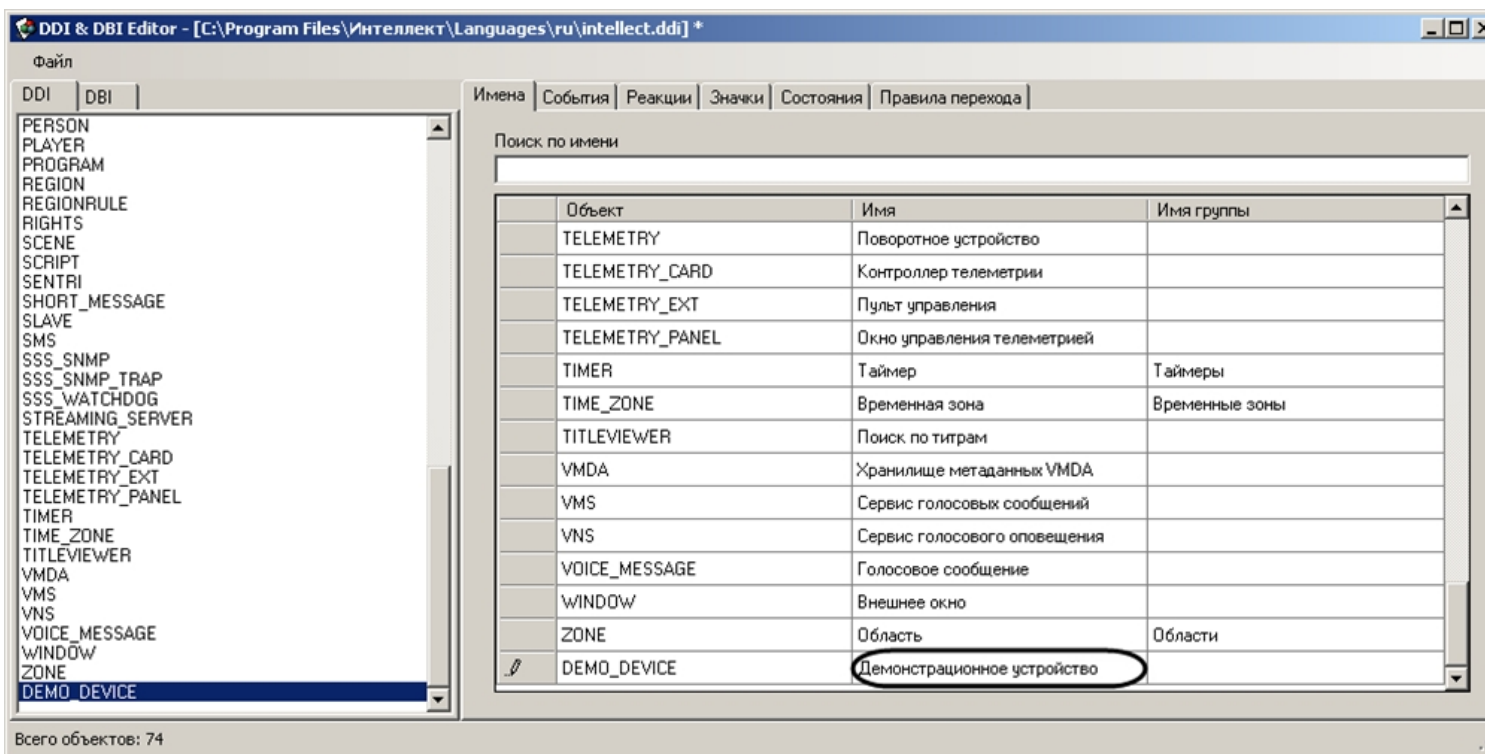
1. Запустить утилиту `ddi.exe`, расположенную в каталоге `Интеллект\Tools`.
2. В окне утилиты перейти на вкладку **DDI**.
3. В меню **Файл** выбрать пункт **Открыть**. В результате выполнения операции появится диалоговое окно **Открыть**.
4. Выбрать файл `intellect.ddi`, расположенный в каталоге `Интеллект\Languages\ru`. В утилите `ddi.exe` отобразится список объектов.
5. Добавить объект, выбрав в контекстном меню списка объектов пункт **Добавить** или нажав кнопку **Insert**.
6. В поле **ID** открывшегося окна ввести имя, используемое для идентификации объекта, и нажать кнопку **OK**.



В результате выполнения операции объект **DEMO\_DEVICE** отобразится в списке объектов.

7. На вкладке **Имена** ввести имя объекта.





8. Добавить информацию об объекте **DEMO\_DEVICE** на соответствующих вкладках.

а. Добавить события **ON** и **OFF** на вкладке **События**.

Имена События Реакции Значки Состояния Правила перехода							
	Название	Описание	Обработка сообщений	Звуковая поддержка	Не слать в сеть	Не протоколировать	Журнал Windows
▶	ON	Включено	▼	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	OFF	Выключено	▼	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
*			▼	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

б. Добавить действия **ON** и **OFF** на вкладке **Реакции**.

Имена События Реакции Значки Состояния Правила перехода			
	Реакция	Описание	Постановка раздела на охрану
	ON	Включить	<input type="checkbox"/>
✎	OFF	Выключить	<input type="checkbox"/>
*			<input type="checkbox"/>

с. На вкладке **Значки** указать часть имени bmp-файла, которая является идентификатором изображения. Идентификатор изображения позволяет использовать несколько bmp-файлов для представления на *Карте* объектов одного типа.

Имена	События	Реакции	Значки	Состояния	Правила перехода
	Имя файла				Название
	demo_device				Модуль DEMO
*					

- d. На вкладке **Состояния** добавить состояния **ON** и **OFF**. Для отображения состояния объекта на *Карте* необходимо указать часть имени, которая является идентификатором состояния, соответствующего bmp-файла.

Имена	События	Реакции	Значки	Состояния	Правила перехода	
	Название			Изображение	Описание	Мерцание при тревоге
	ON			on	Включено	<input type="checkbox"/>
	OFF			off	Выключено	<input type="checkbox"/>
*						<input type="checkbox"/>

**Примечание.**  
Каталог Интеллект\Bmp должен содержать bmp-файлы, имена которых составлены следующим образом:  
**<Идентификатор изображения>\_<Идентификатор состояния>**  
Если идентификатор изображения не задан, то имя bmp-файла должно иметь вид:  
**<Идентификатор объекта>\_<Идентификатор состояния>**

**Примечание**  
Объекты на *Карте* могут быть отображены с помощью линий, т.е. без использования bmp-файлов. В этом случае, если изменяется состояние объекта, меняется цвет линии. Цвет (RGB) состоянию задается следующим образом:  
**<Состояние>\$R:G:B**

- e. На вкладке **Правила перехода** задать правило перехода из одного состояния в другое по определенному событию.

Имена	События	Реакции	Значки	Состояния	Правила перехода
	Событие			Переход из состояния	Переход в состояние
	OFF			ON	OFF
	ON			OFF	ON
*					

**Примечание.**  
Если поле **Переход из состояния** оставить пустым, то переход будет осуществляться из любого состояния.

9. Сохранить изменения, выбрав в меню **Файл** пункт **Сохранить**.

Информация об объекте **DEMO\_DEVICE** внесена.

**Примечание.**  
Поля таблиц утилиты ddi.exe подробно описаны в [ПРИЛОЖЕНИЕ 1. Описание структуры ddi-файла](#)

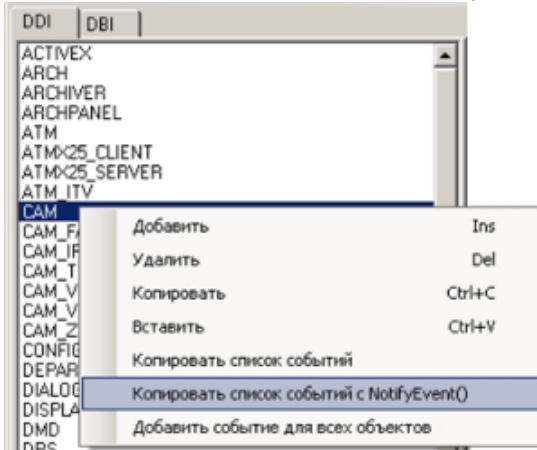
**Внимание!**

После изменения DDI-файлов требуется обновить структуру базы данных с помощью утилиты `idb.exe` (см. шаги 4-7 раздела [Добавление объектов в intellect.dbi](#)).

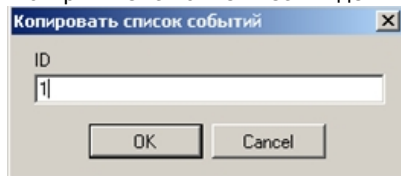
## Дополнительные возможности утилиты `ddi.exe`

Утилита `ddi.exe` представляет собой удобный инструмент для удаления, добавления, редактирования и копирования в буфер обмена свойств объекта (событий, реакций и т.д.). Дополнительно утилита позволяет копировать в буфер обмена события объекта в виде параметра функции `NotifyEvent`. Для этого необходимо выполнить следующие действия:

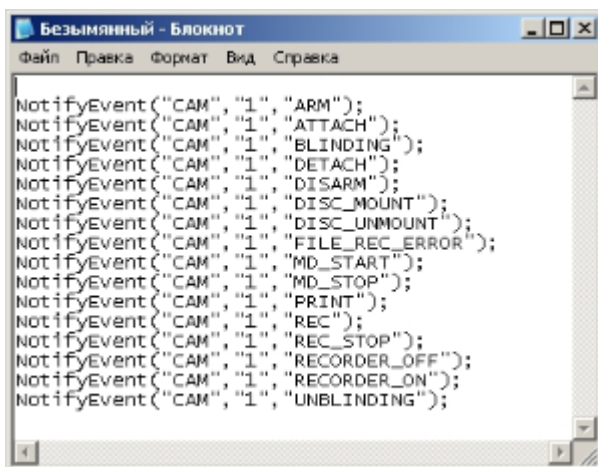
1. В контекстном меню списка объектов выбрать пункт **Копировать список событий с `NotifyEvent()`**.



2. В открывшемся окне ввести идентификационный номер объекта, который следует использовать в функции `NotifyEvent`, и нажать **ОК**.

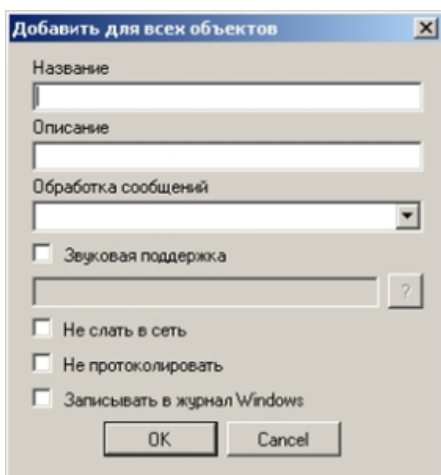


Копирование списка событий завершено. Буфер обмена будет содержать события объекта в виде, представленном на рисунке.



```
NotifyEvent("CAM", "1", "ARM");
NotifyEvent("CAM", "1", "ATTACH");
NotifyEvent("CAM", "1", "BLINDING");
NotifyEvent("CAM", "1", "DETACH");
NotifyEvent("CAM", "1", "DISARM");
NotifyEvent("CAM", "1", "DISC_MOUNT");
NotifyEvent("CAM", "1", "DISC_UNMOUNT");
NotifyEvent("CAM", "1", "FILE_REC_ERROR");
NotifyEvent("CAM", "1", "MD_START");
NotifyEvent("CAM", "1", "MD_STOP");
NotifyEvent("CAM", "1", "PRINT");
NotifyEvent("CAM", "1", "REC");
NotifyEvent("CAM", "1", "REC_STOP");
NotifyEvent("CAM", "1", "RECORDER_OFF");
NotifyEvent("CAM", "1", "RECORDER_ON");
NotifyEvent("CAM", "1", "UNBLINDING");
```

В случае, если требуется добавить событие всем объектам, следует в контекстном меню выбрать пункт **Добавить событие для всех объектов**. В результате выполнения операции будет открыто диалоговое окно **Добавить для всех объектов**, в котором параметрам создаваемого события задаются значения.



Для добавления объектов из других DBI- и DDI-файлов следует в меню **Файл** выбрать пункт **Вставить из файла**.

## Разработка MDL-файла

Для создания mdl-файла необходимо использовать два класса:

1. *NissObjectDLLExt*. Все объекты наследуются от этого класса с переопределением его виртуальных методов.
2. *CoreInterface*. Методы класса используются для получения параметров объектов системы.

Объявленные классы и методы содержатся в заголовочном файле *nissdll.h*. Код, содержащийся в файле *nissdll.h*, представлен в разделе ПРИЛОЖЕНИЕ 2. Объявление классов *NissObjectDLLExt* и *CoreInterface*.

**Примечание.**

Под методами класса подразумеваются процедуры и функции, объявленные в теле класса.

Описание методов класса *NissObjectDLLExt* приведено в таблице.

Метод	Описание	Пример
CoreInterface* m_pCore	Указатель на интерфейс ядра	
virtual BOOL IsWantAllEvents()	Возвращает TRUE, если необходимо в функции OnEvent получать события от всех объектов, FALSE - если только от своего объекта.	если указать "CAM,GRABBER", то при изменении настроек этих объектов для объекта DEMO придут сообщения:  DEMO 1 UPDATE_CAM параметры камеры
virtual CString DescribeSubscribeObjectsList()	Через запятую указываются типы объектов, при изменении которых происходит уведомление текущего объекта	DEMO 1 UPDATE_GRABBER параметры устройства видеоввода
virtual CString GetObjectType()	Возвращает тип объекта	<pre>virtual CString GetObjectType() { return "DEMO"; }</pre>
virtual CString GetParentType()	Возвращает тип родительского объекта	<pre>virtual CString GetParentType() { return "SLAVE"; }</pre>
virtual int GetPos()	Возвращает позицию объекта в ключевом файле "intellect.sec".  <b>Внимание! Этот параметр должен быть согласован с компанией «Ай Ти Ви групп»</b>	<pre>virtual int GetPos() { return -1; }</pre>

*Примечание. При запуске ПК Интеллект в демо-режиме функция возвращает -1*

virtual CString GetPort()	<p>Возвращает номер порта, через который будет происходить соединение и обмен сообщениями между объектом и ядром</p> <p><b>Внимание! Этот параметр должен быть согласован с компанией «Ай Ти Ви групп»</b></p>	<pre>virtual CString GetPort() { return "1100"; }</pre>
virtual CString GetProcessName()	<p>Возвращает имя процесса. Используется ядром для поиска и автоматического запуска исполнительного модуля при старте системы и инициализации модуля</p>	<pre>virtual CString GetProcessName() { return "demo"; }</pre>
virtual CString GetDeviceType()	<p>Определяет тип объекта и характер его поведения.</p> <p>ACD – объект этого типа получает все события, связанные с созданием, изменением и удалением следующих объектов: <b>Пользователи, Временная зона и Уровни доступа</b></p> <p>ACD2– тип аналогичный ACD с дополнительной (обеспеченной ядром) возможностью автоматически удалять временные (с ограниченным сроком действия) карточки</p> <p>Тип ACR указывает на то, что объект является считывателем</p>	<p>Все объекты типа ACR доступны в раскрывающемся списке <b>Точка прохода</b></p>

virtual BOOL HasChild()	Возвращает TRUE, если у объекта есть дочерние объекты, иначе – FALSE	<pre>virtual BOOL HasChild() { return TRUE; }</pre>
virtual UINT HasSetupPanel()	Если у объекта имеется панель настроек, метод возвращает TRUE, иначе – FALSE	<pre>virtual UINT HasSetupPanel() { return TRUE; }</pre>
virtual void OnPanelInit(CWnd*)	Используется при инициализации панели настроек объекта. В качестве параметра передается указатель на окно панели настроек	
virtual void OnPanelLoad(CWnd*,Msg&)	Используется при загрузке панели настроек для получения параметров объекта. В качестве параметра метода передается указатель на окно панели настроек и ссылка на сообщение, через которое осуществляется передача параметров объекта, и заполнение ими необходимых полей в панели настроек	<pre>virtual void OnPanelLoad(CWnd* pwnd,Msg&amp; params) { CString s; s = arams.GetParam("port"); pwnd-&gt;GetDlgItem(IDC_PORT)-&gt; SetWindowText(s); }</pre>

<p>virtual void OnPanelSave(CWnd*,Msg&amp;)</p>	<p>Используется при выгрузке панели настроек для сохранения параметров объекта. В качестве параметра метода передается указатель на окно панели настроек и ссылка на сообщение, через которое осуществляется передача параметров объекта и сохранение их в БД</p>	<pre>virtual void OnPanelSave(CWnd* pwnd,Msg&amp; params) { CString s; pwnd-&gt; GetDlgItem(IDC_PORT)-&gt; GetWindowText(s); params.SetParam("port",s); }</pre>
<p>virtual void OnPanelExit(CWnd*)</p>	<p>Используется при закрытии панели настроек объекта. В качестве параметра передается указатель на окно панели настроек</p>	
<p>virtual void OnPanelButtonPressed(CWnd*,UINT)</p>	<p>Предназначен для обработки нажатий кнопок на панели настроек объекта. В качестве параметра передается указатель на окно панели настроек и идентификатор кнопки.</p> <p><i>Примечание. Числовые значения идентификаторов кнопок должны начинаться с номера <b>1151</b>. Например, для кнопки <b>Test</b> идентификатор в файле <i>Resource.h</i> определяется как:</i></p> <p><b>#define IDC_TEST 1151</b></p>	<pre>Virtual void OnPanelButtonPressed (CWnd* pwnd,UINT id) { if(id==IDC_TEST) { React react("DEMO",Id,"TEST"); m_pCore-&gt;DoReact(react); } }</pre>



	<p>Если необходимо по нажатию кнопки вывести собственное диалоговое окно, созданное в этом же MDL-файле, то необходимо предварительно использовать код, указанный в примере</p>	<pre>HINSTANCE prev_hinst = AfxGetResourceHandle(); HMODULE hRes = GetModuleHandle("demo.mdl"); If (hRes) AfxSetResourceHandle (hRes);  //Код вывода диалогового окна:  CXXXDialog dlg;  dlg.DoModal();  AfxSetResourceHandle(prev_hinst);</pre>
<p>virtual BOOL IsRegionObject()</p>	<p>Указывает на то, что объект будет поддерживать разделы ПК <i>Интеллект</i>. Разделы применяются для группировки объектов. Также они могут использоваться в системе отчетов</p>	
<p>virtual BOOL IsProcessObject()</p>	<p>Указывает на то, что объект будет поддерживать возможность одновременного запуска и параллельной работы нескольких исполнительных модулей. Например, это может использоваться для запуска отдельного модуля непосредственно для каждого COM-порта.</p> <p><i>Примечание. Рекомендуется использовать один рабочий модуль. Это упростит отладку и модификацию модуля</i></p>	

<pre>virtual void OnCreate(Msg&amp;)</pre>	<p>Используется при создании объекта. В качестве параметра передается ссылка на сообщение, содержащее информацию по объекту. Здесь же указываются параметры по умолчанию</p>	<pre>virtual void OnCreate (Msg&amp; msg) { msg.SetParam ("port","COM1"); }</pre>
<pre>virtual void OnInit(Msg&amp;)</pre>	<p>Используется при инициализации объекта. В качестве параметра передается ссылка на сообщение, содержащее информацию по объекту</p>	<pre>virtual void OnInit (Msg&amp; msg) { OnChange (msg, msg); }</pre>
<pre>virtual void OnChange(Msg&amp;,Msg&amp;)</pre>	<p>Используется при изменении объекта. В качестве параметра передаются ссылки на сообщения, содержащие информацию по объекту до и после изменения соответственно</p>	<pre>virtual void OnChange(Msg&amp; msg, Msg&amp; prev) { React react (msg.GetSourceType(), msg.GetSourceId(),"INIT"); react.SetParam("port",msg.GetParam("port")); m_pCore-&gt;DoReact(react); }</pre>

virtual void OnDelete(Msg&)	Используется при удалении объекта. В качестве параметра передается ссылка на сообщение, содержащее информацию по объекту	<pre>virtual void OnDelete (Msg&amp; msg) { React react (msg.GetSourceType(), msg.GetSourceId(),"EXIT"); m_pCore-&gt; DoReact(react); }</pre>
virtual void OnEnable(Msg&)	Предназначен для обработки нажатия кнопки <b>Disable</b> на панели ПК <i>Интеллект</i> при включении объекта. В качестве параметра передается ссылка на сообщение, содержащая информацию по объекту	
virtual void OnDisable(Msg&)	Предназначен для обработки нажатия кнопки <b>Disable</b> на панели ПК <i>Интеллект</i> при выключении объекта. В качестве параметра передается ссылка на сообщение, содержащая информацию по объекту	
virtual BOOL OnEvent(Event&)	Служит для обработки событий, передаваемых в качестве параметра	<pre>virtual BOOL OnEvent(Event&amp; event) { If (event.GetAction() == "ACCESS_IN"    event.GetAction() == "ACCESS_OUT") {</pre>

```
Msg per = m_pCore-> FindPersonInfoByCard(event.GetParam("facility_code"),
event.GetParam("card"));

event.SetParam
("param0", !per.GetSourceId().IsEmpty() ?
per.GetParam("name") : event.GetParam("facility_code") + event.GetParam("card"));
event.SetParam("param1", per.GetSourceId() );
}

Else

If (event.GetAction() == "NOACCESS_CARD")
{
event.SetParam
("param0",event.GetParam("facility_code") + event.GetParam("card"));
}
}
```

		<pre> return TRUE; } </pre>
virtual BOOL OnReact(React&)	Служит для обработки реакций, передаваемых в качестве параметра	

В глобальной функции *CreateNissObject(CoreInterface\* core)* необходимо создать экземпляры описанных объектов, поместить их в массив *CNissObjectDLLExtArray* и вернуть указатель на объект этого массива. Через эту функцию необходимо получить указатель на интерфейс ядра, который в дальнейшем используется в объектах для обращения к методам данного интерфейса:

```

CNissObjectDLLExtArray* APIENTRY CreateNissObject(CoreInterface* core)
{
    CNissObjectDLLExtArray* ar = new CNissObjectDLLExtArray;

    ar->Add(new NissObjectDemo(core));
    ar->Add(new NissObjectDemoDevice(core));

    return ar;
}

```

После загрузки DLL-файла ядро вызывает функцию *CreateNissObject* и получает указатели на все используемые объекты.

Все панели настроек объектов хранятся в ресурсах в виде диалогов. Идентификаторы диалогов строятся по схеме **IDD\_object\_SETUP**, где **object** – это имя соответствующего объекта. Например, для объекта **DEMO** – это **IDD\_DEMO\_SETUP**, а для объекта **DEMO\_DEVICE** – это **IDD\_DEMO\_DEVICE\_SETUP**.

**i** **Примечание.**  
Для того чтобы в дереве настроек у объекта отображался свой собственный значок, необходимо в ресурсах DLL-файла создать **BITMAP** размером 14x14 с именем объекта.

## Мастер создания MDL-файла

Для автоматизации процесса создания MDL-файла используется `intellect_md1.awx` (см. каталог *Wizard*).

Создание MDL-файла с помощью Мастера выполняется следующим образом:

1. Поместить `intellect_md1.awx` в каталог `Program Files\Microsoft Visual Studio\Common\MSDev98\Template`.
2. Запустить *Microsoft Visual C++*.
3. Создать новый проект **INTELLECT MDL WIZARD**.
4. Выполнить настройку проекта, следуя предложенным шагам.

В результате будет создан шаблон системного объекта. Проект будет включать все необходимые файлы, в том числе файл с описанием структуры объекта для `intellect.dbi`.



**Внимание!**

В настройках проекта требуется заменить расширение результирующего файла с `dll` на `mdl`.



**Примечание.**

При сборке проекта необходимо использовать **Release**.

## Разработка RUN-файла

Управление устройствами выполняется через обмен сообщениями (командами) между RUN-файлом и ядром системы. Для реализации данного взаимодействия программного модуля с ядром используется *IIDK*, подробно рассмотренный в разделе *Intellect Integration Developer Kit (IIDK)*. Необходимую информацию можно также почерпнуть из исходных файлов демонстрационного модуля, которые прилагаются к документации.

Ниже приведен пример использования средств разработки *IIDK* для демонстрационного модуля *DEMO*.

```

CString port = "1100";
CString ip = "127.0.0.1";
CString id = "";
BOOL IsConnect = Connect (ip, port, id, myfunc);
if (!IsConnect)
{
// не удалось подключиться
AfxMessageBox("Error");
Return;
}
SendMsg(id,"CAM|1|REC"); // поставить камеру 1 на запись
SendMsg(id, "DEMO|1|RESTORE"); // восстановление связи с объектом DEMO
//включить устройство DEMO_DEVICE с адресом 1
SendMsg(id,"DEMO_DEVICE|1|ON|params<1>,param0_name<address>,param0_val<1>");
Disconnect(id);

```

 **Внимание!**  
Если создан mtl-файл, то для подключения к ядру ПК *Интеллект* объект **Интерфейс IIDK** в системе не создается. В качестве идентификатора подключения передается пустая строка, то есть id равен "".

При выгрузке модуля ему посылается сообщение **WM\_EXIT**:

**#define WM\_EXIT (WM\_USER+2000)**

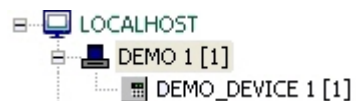
Используя функцию WinAPI – *PostThreadMessage*, необходимо перехватить это сообщение и обеспечить корректную выгрузку модуля. В VC++ и MFC сообщение **WM\_EXIT** отлавливается в классе, наследуемом от *CWinApp*, в Delphi и CBuilder – *TApplication*.

## Создание и настройка интегрированных объектов (модулей) в ПК Интеллект

Для создания и настройки интегрированных объектов (модулей) в ПК *Интеллект* необходимо выполнить следующие действия:

1. Разместить MDL и RUN-файлы в каталоге *Интеллект\Modules*.
2. Запустить ПК *Интеллект*.
3. На базе объекта **Компьютер** создать добавленные с помощью программного модуля объекты. Для демонстрационного модуля *DEMO* необходимо на базе объекта **Компьютер** создать объект **DEMO**.

**Примечание.**  
На базе объекта **DEMO** создается дочерний объект **DEMO\_DEVICE**.



В результате выполнения операции будут доступны панели настроек объектов.  
Панель настроек объекта DEMO:



1 DEMO 1

Компьютер  Отключить

LOCALHOST

COM1 Port

Test

Применить Отменить

Панель настроек объекта DEMO\_DEVICE:

1 DEMO\_DEVICE 1  
DEMO  Отключить  
DEMO 1  
1 Address  
Применить Отменить

4. Произвести настройку объектов.

Процесс создания и настройки интегрированных объектов в ПК *Интеллект* завершен.

## Элемент управления ActiveX CamMonitor.ocx

### Общее описание ActiveX-компонента CamMonitor.ocx

#### На странице:

- [Общее описание CamMonitor.ocx](#)
- [Требования к разработчику](#)

#### Общее описание CamMonitor.ocx

CamMonitor.ocx представляет собой ActiveX компонент, который является полным аналогом интерфейсного объекта **Монитор видеонаблюдения**. Он позволяет управлять камерами, просматривать архив и т.д.

Компонент CamMonitor.ocx поддерживает работу в демо-режиме.

#### Требования к разработчику

Для использования CamMonitor.ocx требуется:

1. знание любого языка программирования, поддерживающего использование компонентов Component Object Model (COM);
2. знание основ программирования в Win32;
3. наличие среды разработки, поддерживающей работу с осх-файлами.

✔ Требования к программному обеспечению, используемому при интеграции

## Установка CamMonitor.osx

Установка CamMonitor.osx осуществляется при помощи файла CamMonitorInstaller.exe, который располагается в папке <Директория установки *Интеллект*>\Redist.

При установке на 32-битную систему файл cammonitor.osx помещается в папку WINDOWS\system32, при установке на 64-битную систему – в папку WINDOWS\SysWOW64. Также осуществляется его стандартная регистрация как компонента ActiveX.

CamMonitorInstaller.exe выполняет установку требуемых файлов для всех пользователей системы.

Помимо самой библиотеки устанавливается также пакет драйверов CodecPack и утилита ITV VideoPlayer, которая работает с использованием компонента CamMonitor.osx и позволяет просматривать видеоархив по выбранной камере. Данная утилита по умолчанию устанавливается в папку C:\Program Files\ITV VideoPlayer\. Интерфейс утилиты схож с интерфейсом Converter.exe, однако в ней отсутствуют некоторые функции Converter.exe, не связанные с просмотром архива. Данную утилиту можно использовать для проверки корректности установки CamMonitor.osx.

## Параметры CamMonitor.osx

На странице:
<ul style="list-style-type: none"><li>• <a href="#">CamMenuOptions</a></li><li>• <a href="#">CamMenuProcessingOptions</a></li><li>• <a href="#">CamButtonsOptions</a></li><li>• <a href="#">MainPanelOptions</a></li><li>• <a href="#">KeysOptions</a></li><li>• <a href="#">OverlayMode</a></li><li>• <a href="#">Пример использования параметров</a></li></ul>

В этом разделе перечислены параметры, позволяющие настроить режим работы компонента CamMonitor: задать отображаемые элементы интерфейса, а также режим использования оверлея.

Все параметры представляют собой целое число типа long.

Значения параметров для настройки элементов интерфейса, перечисленные в таблицах, сформированы таким образом, чтобы в двоичном представлении числа была только одна единица. Чтобы задать значение параметра, необходимо при помощи операции исключающего ИЛИ (XOR) объединить требуемые значения параметров, получив таким образом некое число, разряды которого в двоичном представлении говорят о том, какие элементы интерфейса следует отображать, а какие нет. См. также [Пример использования параметров](#).

Параметр OverlayMode отличается от прочих: он принимает значения от 0 до 2, и его значение задает режим использования оверлея.

### CamMenuOptions

**CamMenuOptions** : long

Позволяет настроить функциональное меню камеры.

Допускается установка одного или нескольких флагов.

Возможные значения:

Значение	Описание
#define MENU_ENABLE_OPTION 0x00000001	Отображать меню
#define MENU_ARM_ENABLE_OPTION 0x00000002	Отображать пункт меню <b>Поставить на охрану</b>
#define MENU_REC_ENABLE_OPTION 0x00000004	Отображать пункт меню <b>Начать запись</b>
#define MENU_CAMS_ENABLE_OPTION 0x00000008	Отображать пункт меню <b>Камера</b>
#define MENU_TITLES_ENABLE_OPTION 0x00000010	Отображать пункт меню <b>Отображение титров</b>
#define MENU_PROCESSING_ENABLE_OPTION 0x00000020	Отображать пункт меню <b>Обработка</b>
#define MENU_EXPORT_ENABLE_OPTION 0x00000040	Отображать пункт меню <b>Экспорт</b>

## CamMenuProcessingOptions

**CamMenuProcessingOptions** : long

Позволяет настроить меню **Обработка** в функциональном меню камеры.

Допускается установка одного или нескольких флагов.

Возможные значения:

Значение	Описание
#define MENU_PROCESSING_DEINTERLACE_ENABLE_OPTION 0x00000001	Отображать пункт меню <b>Деинтерлейсинг</b>
#define MENU_PROCESSING_ZOOM_ENABLE_OPTION 0x00000002	Отображать пункт меню <b>Увеличение</b>
#define MENU_PROCESSING_CONTRAST_ENABLE_OPTION 0x00000004	Отображать пункт меню <b>Контрастирование</b>
#define MENU_PROCESSING_MASK_ENABLE_OPTION 0x00000008	Отображать пункт меню <b>Маска детектора</b>

## CamButtonsOptions

**CamButtonsOptions** : long

Позволяет настроить отображение кнопок компонента CamMonitor.

Допускается установка одного или нескольких флагов.

Возможные значения:

Значение	Описание
#define BUTTON_ARCH_ENABLE_OPTION 0x00000001	Отображать кнопку входа в архив
#define BUTTON_TIME_ENABLE_OPTION 0x00000002	Отображать время
#define BUTTON_NAME_ENABLE_OPTION 0x00000004	Отображать название камеры
#define BUTTON_MENU_ENABLE_OPTION 0x00000008	Отображать кнопку вызова функционального меню
#define BUTTON_RAYS_ENABLE_OPTION 0x00000010	Не используется
#define BUTTON_MICS_ENABLE_OPTION 0x00000020	Не используется

## MainPanelOptions

**MainPanelOptions** : long

Позволяет настроить отображение панели инструментов CamMonitor.

Допускается установка одного или нескольких флагов.

Возможные значения:

Значение	Описание
#define MAIN_PANEL_ENABLE_OPTION 0x00000001	Включает отображение панели

## KeysOptions

**KeysOptions** : long

Позволяет настроить управление компонентом при помощи клавиатуры и мыши.

Допускается установка одного или нескольких флагов.

Возможные значения:

Значение	Описание
#define TELEMETRY_DISABLE_OPTION 0x00000001	Отключает возможность управления компонентом CamMonitor при помощи горячих клавиш, доступных для Монитора видеонаблюдения (см. <a href="#">Монитор видеонаблюдения</a> ).
#define TELEMETRY_DISABLE_OPTION 0x00000002	Отключает возможность управления телеметрией из компонента CamMonitor (см. <a href="#">Управление поворотными устройствами</a> )

## OverlayMode

**OverlayMode** : long

Задает режим отображения.

Возможные значения:

Значение	Описание
0	Не использовать Overlay
1	Overlay 1
2	Overlay 2

## Пример использования параметров

```
DWORD options = CamMonitor1->CamMenuOptions;  
  
options = options^MENU_CAMS_ENABLE_OPTION^MENU_ARM_ENABLE_OPTION^MENU_REC_ENABLE_OPTION;  
  
CamMonitor1->CamMenuOptions = options;  
  
CamMonitor1->CamMenuProcessingOptions ^= MENU_PROCESSING_MASK_ENABLE_OPTION;
```

## Методы CamMonitor.осх

### На странице:

- [Connect](#)
- [ShowCam](#)
- [DoReactMonitor](#)
- [RemoveAllCams](#)
- [IsConnected](#)
- [GetCurIp](#)
- [SendRawMessage](#)
- [Disconnect](#)
- [SetCallBackOptions](#)

## Connect

**Connect**(BSTR **ip**, BSTR **login**, BSTR **password**, BSTR **arch\_password**, long **param**) установка соединения с сервером.

- BSTR **ip** – IP адрес сервера.

- BSTR **login** – логин для соединения с сервером (может быть пустым).
- BSTR **password** – пароль на соединение с сервером (может быть пустым).
- BSTR **arch\_password** – пароль для доступа к архиву (т.е. пароль администратора, может быть пустым).
- long **param** – роль, исполняемая сервером:
  - 0 – видеосервер;
  - 1 – оперативный архив;
  - 2 – видеошлюз.

Установка связи с сервером происходит **асинхронно**.

## ShowCam

**ShowCam**(long **cam\_id**, long **compress**, long **show**) выводит/скрывает камеру с экрана

- long **cam\_id** – идентификатор(номер) камеры.
- long **compress** – уровень компрессии видео 0-5 (для локальной камеры =0). -1 - отображает в оригинальном формате транслируемом с камеры без рекомпрессии.
- long **show** – флаг означающий действие: показать/скрыть камеру (1/0).

## DoReactMonitor

**DoReactMonitor**(BSTR **react\_string**) – управление поведением монитора/камер

- BSTR **react\_string** – строковое представление реакции.

**Пример формирования react\_string:**

```
react_string = "MONITOR||ARCH_FRAME_TIME|cam<3>,date<dd-mm-yy>,time<hh:mm:ss>,mode<1>";
CamMonitor1->DoReactMonitor(react_string);
```

**Результат** вызова функции с таким параметром: камера 3 будет переведена в режим архива видеосервера, и архив будет позиционирован на дату «dd-mm-yy» и время «hh:m m:ss» (дату и время необходимо задавать только в таком формате). Параметр mode может принимать следующие значения:

- 0 – видеошлюз, если задан (если не задан, то видеосервер).
- 1 – видеосервер.
- 2 – долговременный архив.

"MONITOR|<id игнорируется >|ARCH\_FRAME\_TIME|..."



### Примечание:

Опционально можно указать также точность при позиционировании до миллисекунд например:

```
DoReactMonitor("MONITOR||ARCH_FRAME_TIME|cam<3>,date<02-10-05>,time<12:12:22.345>,mode<1>")
```

## RemoveAllCams

**RemoveAllCams()** : **long** – удаление всех камер с экрана

## IsConnected

**IsConnected()** : **boolean** – метод говорит о наличии/отсутствии связи с видеосервером.

## GetCurIp

**GetCurIp()** : **BSTR** – возвращает IP адрес сервера, ранее указанный при вызове **Connect**.

## SendRawMessage

**SendRawMessage(BSTR msg)** – дать видеосерверу команду на исполнение.

- **BSTR msg** – строковое представление команды.

**Примеры вызова функции:**

```
m_Cam.SendRawMessage("CAM|1|REC");
```

```
m_Cam.SendRawMessage("CAM|1|REC_STOP");
```

```
m_Cam.SendRawMessage("CAM|1|ARM");
```

```
m_Cam.SendRawMessage("CAM|1|DISARM");
```

## Disconnect

**Disconnect()** – осуществляет отсоединение от видеосервера.

## SetCallbackOptions

**SetCallbackOptions(int cam\_id, int options)** – задает параметры получения видеоизображения с камеры.



- int **cam\_id** – идентификатор(номер) камеры.
- int **options** – опции. Возможные значения параметра **options**:
  - WithoutVideoFrame = 0x00 – не присылать кадры из модуля видео.
  - WithVideoFrame = 0x01 – присылать кадры из модуля видео.
  - WithExtendedParams = 0x02 – получать видео кадры с дополнительными параметрами (время, число кадров, субтитры).
  - WithInformationLayout = 0x04 – отображать видео в окне с элементами управления (контекстное меню).
  - WithCompressedData = 0x08 – отображать видео в исходном формате без распаковки (если оно есть).
  - WithoutDecode = 0x10 – отключить декодирование видео на сервере.
  - WithoutSubtitles=0x20 – отключить субтитры.



**Примечание.**

Параметр options формируется так же, как параметры компонента CamMonitor.oscx – см. [Параметры CamMonitor.oscx](#)

## События CamMonitor.oscx

**OnCamListChange** (long **cam\_id**, long **action**) - возникает при установлении связи с сервером или при изменении количества камер на сервере

- long **cam\_id** – идентификатор камеры.
- long **action** – равен 1, если камера с **id** == **cam\_id** существует, иначе **action** == 0.

Данное событие возникает столько раз, сколько камер на данном видеосервере. Признаком окончания вызовов **OnCamListChange** является отрицательное значение параметра **cam\_id** (cam\_id < 0).

Если на сервере находится, например, 3 камеры (1, 2, 3), то последовательно возникнут события:

CamListChange(1,1)

CamListChange(2,1)

CamListChange(3,1)

CamListChange(-1,1)

**Пример:**

Показать камеру с cam\_id =2 с уровнем компрессии compress =1;

```
CamMonitor1CamListChange(long cam_id, long action)
{
  if(cam_id == -1)
  {
    CamMonitor1->ShowCam(2,1,1);
  }
}
```

## HTTP API ПК Интеллект

# Общие сведения о HTTP API

Программно HTTP API предоставляется модулем web2 (*Веб-сервер 2.0*).



## Примечание.

См. [Руководство Администратора](#), раздел [Настройка Сервера для подключения Клиентов с помощью модуля Веб-сервер 2.0](#).

HTTP API позволяет использовать следующие функции:

1. Получение сведений об интерактивных картах: списка карт, имени карты, списка слоев карты, параметров слоя, фонового рисунка слоя, информации о списке точек и отдельной точке на слое (см. [Карта](#)).
2. Получение сведений о классах объектов, созданных на Сервере, списка состояний для класса объектов и информации о состоянии, иконки для определенного состояния (см. [Классы объектов](#)).
3. Получение списка объектов, созданных на сервере, информации об отдельном объекте, состоянии объекта, списка доступных действий с объектом (см. [Объекты](#)).
4. Получение событий с Сервера как отдельно, так и блоками (см. [Получение событий](#)).
5. Отсылать команды на Сервер (см. [Отсылка команд на сервер](#)).
6. Запускать выполнение макрокоманд (см. [Макрокоманды](#)).
7. Работать с видео: получить кадры, запрашивать конфигурацию, получать живое видео и архив, управлять записью, ставить и снимать камеры с охраны, управлять телеметрией (см. [Видео](#)).
8. Использовать системы нотификации для подписки приложения на сообщения APNS (см. [Нотификация](#)).
9. Получать живой и архивный звук (см. [Звук](#)).
10. Отправлять события и реакции в ядро ПК Интеллект (см. [Отправка реакций и событий в ПК Интеллект по HTTP-запросу](#)).

В примерах, приводимых в данном разделе, используются следующие обозначения:

- Port – порт.
- /web2 – веб-контекст, в котором работает модуль web2. Это контекст веб-приложения.

Далее описание будет опускаться там, где действие запроса понятно из контекста.



## Внимание!

URL, id объектов и расширения файлов чувствительны к регистру.



## Примечание.

Дата и время указываются в формате RFC3339, подробнее см. <http://www.ietf.org/rfc/rfc3339.txt>

## Версия продукта

Для идентификации сервера можно использовать URL

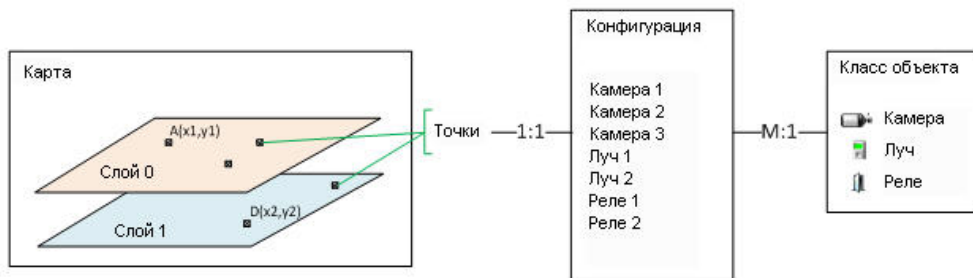
`http://example.com:[port]/web2/product/version`

Если в ответ приходит text/plain строка типа

Intellect 4.8.4

Это означает, что сервер поддерживает протокол, описанный в данном документе. Строка может меняться в зависимости от версии продукта. Это сделано для того, чтобы различать два схожих по функциональным возможностям, но разных по протоколу веб-сервера в разных продуктах.

# Карта



На Сервере может быть создано несколько карт. Каждая карта может содержать один и более слоёв. На каждом слое расположены точки. Каждая точка связана с одним из объектов конфигурации.

Конфигурация – это объекты ПК *Интеллект*. Каждый объект является объектом определённого класса. Каждый объект имеет одно состояние и список действий, которые можно с ним производить.

Класс объекта описывает его вид (значки), возможные состояния и возможные действия с объектом в каждом из состояний .

## Получение списка карт

Карт может быть 0 и более.

[http://example.com:\[port\]/web2/secure/kartas/](http://example.com:[port]/web2/secure/kartas/)

### Пример ответа:

```
<kartas>
<karta>
<id>plan</id>
<name>This is plan of a building</name>
</karta>
<karta>
<id>site</id>
<name>This is site around the building</name>
</karta>
</kartas>
```

## Информация об одной карте

plan – id карты.

http://example.com:[port]/web2/secure/kartas/**plan**/

**Пример ответа:**

```
<karta>
<id>plan</id>
<name>This is plan of a building</name>
</karta>
```

## Список слоёв для выбранной карты

plan – id карты.

Слоёв может быть 1 и более.

http://example.com:[port]/web2/secure/kartas/**plan**/layers/

**Пример ответа:**

```
<layers>
<layer>
<height>1000</height>
<id>base</id>
<mapId>plan</mapId>
<name>Base layer for plan</name>
<width>1000</width>
<zoomDef>1.0</zoomDef>
<zoomMax>4.0</zoomMax>
<zoomMin>0.25</zoomMin>
<zoomStep>0.25</zoomStep>
</layer>
</layers>
```

**Описание параметров:**

Height – высота картинки слоя в пикселях;

Width – ширина картинки слоя в пикселях;

zoomMin – минимальный масштаб картинки;

zoomMax – максимальный масштаб картинки;

zoomStep – шаг увеличения масштаба при zoom in и zoom out;

zoomDef – масштаб по-умолчанию.

**Пример.** Пусть ширина картинки равна 100 пикселям. Тогда ширина для масштаба 0,25 будет  $100 * 0,25 = 25$  пикселей.

## Информация о конкретном слое

Описание параметров см. в разделе [Список слоёв для выбранной карты](#).

base – id слоя.

[http://example.com:\[port\]/web2/secure/kartas/plan/layers/base/](http://example.com:[port]/web2/secure/kartas/plan/layers/base/)

### Пример ответа:

```
<layer>
  <height>1000</height>
  <id>base</id>
  <mapId>plan</mapId>
  <name>Base layer for plan</name>
  <width>1000</width>
  <zoomDef>1.0</zoomDef>
  <zoomMax>4.0</zoomMax>
  <zoomMin>0.25</zoomMin>
  <zoomStep>0.25</zoomStep>
</layer>
```

## Фоновый рисунок слоя

[http://localhost:8080/server-1.0/secure/kartas/plan/layers/base/image.\[png|jpg\]](http://localhost:8080/server-1.0/secure/kartas/plan/layers/base/image.[png|jpg])

В ответ приходит изображение в формате png или jpg.

На запрос JPG, Jpg, JPEG, PNG будет возвращаться ошибка 404

## Список точек на слое

[http://example.com:\[port\]/web2/secure/kartas/plan/layers/base/points/](http://example.com:[port]/web2/secure/kartas/plan/layers/base/points/)

id – совпадает с id объекта из конфигурации. Id не обязательно всегда равен CAM:1. Следует воспринимать id как строку.

Координатная сетка привязана к слою следующим образом:



Т.е. x и y не могут быть отрицательными, но могут быть дробными.

**Пример ответа:**

```
<points>
  <point>
    <id>CAM:1</id>
    <layerId>base</layerId>
    <mapId>plan</mapId>
    <x>100.0</x>
    <y>100.0</y>
  </point>
  <point>
    <id>CAM:2</id>
    <layerId>base</layerId>
    <mapId>plan</mapId>
    <x>200.0</x>
    <y>200.0</y>
  </point>
```

```
<point>
  <id>GRAY:1</id>
  <layerId>base</layerId>
  <mapId>plan</mapId>
  <x>300.0</x>
  <y>300.0</y>
</point>
<point>
  <id>GRAY:2</id>
  <layerId>base</layerId>
  <mapId>plan</mapId>
  <x>400.0</x>
  <y>400.0</y>
</point>
<point>
  <id>GRELE:1</id>
  <layerId>base</layerId>
  <mapId>plan</mapId>
  <x>500.0</x>
  <y>500.0</y>
</point>
<point>
  <id>GRELE:2</id>
  <layerId>base</layerId>
  <mapId>plan</mapId>
  <x>600.0</x>
  <y>600.0</y>
</point>
</points>
```

## Информация об отдельной точке на слое

[http://example.com:\[port\]/web2/secure/kartas/plan/layers/base/points/CAM:2](http://example.com:[port]/web2/secure/kartas/plan/layers/base/points/CAM:2) – запрос информации о точке, соответствующей камере с идентификатором 2.

### Пример ответа:

```
<point>
  <id>CAM:2</id>
  <layerId>base</layerId>
  <mapId>plan</mapId>
  <x>200.0</x>
  <y>200.0</y>
</point>
```

## Классы объектов

### Список классов объектов, которые существуют на сервере

[http://example.com:\[port\]/web2/secure/objectClasses](http://example.com:[port]/web2/secure/objectClasses)

### Пример ответа:

```
<objectClasses>
  <objectClass>
    <id>GRELE</id>
  </objectClass>
  <objectClass>
    <id>USERS</id>
  </objectClass>
  <objectClass>
    <id>CAM</id>
  </objectClass>
  <objectClass>
    <id>RIGHTS</id>
  </objectClass>
```



```
<objectClass>
  <id>GRAY</id>
</objectClass>
</objectClasses>
```

## Отдельный класс объектов

[http://example.com:\[port\]/web2/secure/objectClasses/GRELE/](http://example.com:[port]/web2/secure/objectClasses/GRELE/)

### Пример ответа:

```
<objectClass>
  <id>GRELE</id>
</objectClass>
```

## Список состояний для определённого класса объектов

[http://example.com:\[port\]/web2/secure/objectClasses/GRELE/states/](http://example.com:[port]/web2/secure/objectClasses/GRELE/states/) - получить список состояний для класса объектов **Реле**.

### Пример ответа:

```
<states>
  <state>
    <id>off</id>
  </state>
  <state>
    <id>on</id>
  </state>
  <state>
    <id>disabled</id>
  </state>
</states>
```

## Информация о конкретном состоянии

[http://example.com:\[port\]/web2/secure/objectClasses/\[ObjectClass\]/states/\[State\]/](http://example.com:[port]/web2/secure/objectClasses/[ObjectClass]/states/[State]/)

### Пример:

http://example.com:[port]/web2/secure/objectClasses/**GRELE**/states/**off**/ - получение информации о состоянии OFF класса объектов **Реле**.

### Пример ответа:

```
<state>
<id>off</id>
</state>
```

## Получение иконки для определённого состояния

http://example.com:[port]/web2/secure/objectClasses/**[ObjectClass]**/states/**[State]**/image.png

### Пример:

http://example.com:[port]/web2/secure/objectClasses/**GRELE**/states/**off**/image.png - получение иконки для состояния OFF класса объектов **Реле**.

В ответ приходит изображение в формате png:



## Объекты

### Получение списка объектов

http://example.com:[port]/web2/secure/configuration?pageItems=3&page=2 – запрос возвращает список камер, добавленных в Веб-сервер.

### Параметры:

**page** – необязательный параметр. Задаёт номер страницы, отображаемой в результате запроса. По умолчанию 1.

**pageItems** – необязательный параметр. Задаёт количество объектов, выводимых на странице. По умолчанию 1000.



### Внимание!

Если в системе много объектов (>1000) необходимо использовать постраничный вывод.

Обработка всех объектов производится перебором страниц, до получения пустого массива.

### JSON

```
[ {
  "type" : "CAM",
  "id" : "CAM:2",
  "extId" : "2",
  "name" : "Camera 2",
  "regionId" : "2.1",
  "state" : {
    "id" : "alarmed",
```

```

    "type" : "ALARM"
  },
  "presets" : [ ]
}, {
  "type" : "CAM",
  "id" : "CAM:1",
  "extId" : "1",
  "name" : "Camera 1",
  "state" : {
    "id" : "armed",
    "type" : "NORMAL"
  },
  "presets" : [ ]
}, {
  "type" : "GRAY",
  "id" : "GRAY:1",
  "extId" : "1",
  "name" : "Sensor 1",
  "state" : {
    "id" : "disconnected",
    "type" : "ALARM"
  }
}, {
  "type" : "GRELE",
  "id" : "GRELE:2",
  "extId" : "2",
  "name" : "Relay 2",
  "state" : {
    "id" : "disabled",
    "type" : "NORMAL"
  }
}, {
  "type" : "GRELE",
  "id" : "GRELE:1",
  "extId" : "1",
  "name" : "Relay 1",
  "regionId" : "2.1",
  "state" : {
    "id" : "disabled",
    "type" : "NORMAL"
  }
}, {
  "type" : "GRAY",
  "id" : "GRAY:2",
  "extId" : "2",
  "name" : "Sensor 2",
  "state" : {
    "id" : "disconnected",

```

```
    "type" : "ALARM"
  }
} ]
```

## Информация об отдельном объекте

[http://example.com:\[port\]/web2/secure/configuration/GRAY:2/](http://example.com:[port]/web2/secure/configuration/GRAY:2/) – получение информации об объекте **Луч** с идентификатором 2.

### Пример ответа:

```
<GRAY>
  <id>GRAY:2</id>
  <name>Луч 2</name>
  <state>
    <id>alarmed</id>
  </state>
</GRAY>
```

## Состояние отдельного объекта

[http://example.com:\[port\]/web2/secure/configuration/GRAY:2/state/](http://example.com:[port]/web2/secure/configuration/GRAY:2/state/) – получение состояния объекта **Луч** с идентификатором 2.

### Пример ответа:

```
<state>
<fullState>ON,ARMED</fullState>
<id>armed</id>
<type>NORMAL</type>
</state>
```

В теге `<fullState>` передается полное состояние объекта, хранящееся в базе данных, а в тегах `<type>` и `<id>` – состояние объекта в терминах HTTP API. Возможные значения данных параметров для луча:

Состояние луча	Полный список состояний
Луч снят с охраны	<code>&lt;fullState&gt;DISARMED&lt;/fullState&gt;</code> <code>&lt;id&gt;connected&lt;/id&gt;</code> <code>&lt;type&gt;NORMAL&lt;/type&gt;</code>
Луч снят с охраны + тревога	<code>&lt;fullState&gt;ALARMED&lt;/fullState&gt;</code> <code>&lt;id&gt;alarmed&lt;/id&gt;</code> <code>&lt;type&gt;ALARM&lt;/type&gt;</code>

Луч снят с охраны + тревога принята	<fullState>CONFIRMED</fullState> <id>confirmed</id> <type>NORMAL</type>
Луч снят с охраны + замкнут	<fullState>ON,DISARMED</fullState> <id>connected</id> <type>NORMAL</type>
Луч снят с охраны + разомкнут	<fullState>DISARMED</fullState> <id>connected</id> <type>NORMAL</type>
Луч снят с охраны + потеря сигнала	<fullState>DISARMED,DETACHED_DISARM</fullState> <id>disconnected</id> <type>ALARM</type>
Луч на охране	<fullState>ARMED</fullState> <id>armed</id> <type>NORMAL</type>
Луч на охране + тревога	<fullState>ALARMED</fullState> <id>alarmed</id> <type>ALARM</type>
Луч на охране + тревога принята	<fullState>CONFIRMED</fullState> <id>confirmed</id> <type>NORMAL</type>
Луч на охране + замкнут	<fullState>ON,ARMED</fullState> <id>armed</id> <type>NORMAL</type>
Луч на охране + разомкнут	<fullState>ARMED</fullState> <id>armed</id> <type>NORMAL</type>
Луч на охране + потеря сигнала	<fullState>DETACHED_DISARM</fullState> <id>disconnected</id> <type>ALARM</type>

## Список доступных действий с объектом, находящимся в определённом состоянии

Список действий запрашивается не по классу объекта, а берётся из контекста конкретного объекта, т.к. возможны различные права пользователя на объекты одного и того же

класса.

Работа с полученным списком описана в разделе [Отсылка команд на сервер](#).

`http://example.com:[port]/web2/secure/configuration/GRAY:2/state/actions/` – получение списка доступных действия для объекта **Луч** с идентификатором 2.

**Пример ответа:**

```
<actions>
  <action>
    <description>Disarm ray</description>
    <id>ray.disarm</id>
  </action>
  <action>
    <description>Confirm alarm</description>
    <id>ray.confirm</id>
  </action>
</actions>
```

Если состояние объекта не предусматривает никаких действий, то xml будет таким:

```
<actions/>
```

## Получение событий

Соединение не разрывается и события приходят бесконечно.

action – тип события. Возможные значение: create, delete, update.

Все поля ниже опциональны:

objectId – id объекта, от которого приходит событие (обязательно приходит с update, delete, create).

state – id нового состояния объекта (обязательно приходит в create. Если состояние не изменилось, то в событии update состояния не будет).

x, y – новые координаты, если изменились.

**Запрос:**

`http://example.com:[port]/web2/secure/feed/`

**Примеры ответа:**

```
<message>
```

```
<action>update</action>
<objectId>CAM:1</objectId>
<state>disconnected</state>
</message>
```

```
<message>
  <action>state</action>
  <objectId>CAM:1</objectId>
  <x>10.0</x>
  <y>123.9</y>
</message>
```

```
<message>
  <action>state</action>
  <objectId>CAM:1</objectId>
  <state>connected</state>
  <x>300.8</x>
  <y>670</y>
</message>
```

```
<message>
  <action>state</action>
  <objectId>CAM:1</objectId>
  <x>100</x>
  <y>100</y>
</message>
```

```
<message>
  <action>ping</action>
```

</message>

## Получение событий видеоподсистемы блоками

http://example.com:[port]/web2/secure/events/

### Параметры:

**from** – Самая старая дата промежутка поиска сообщений. (2012-12-27T15%3A19%3A16.000%2B04%3A00)

**to** – Самая последняя дата промежутка поиска сообщений. (2012-12-27T15%3A19%3A16.000%2B04%3A00)

**count** – максимальное количество сообщений в ответе [1, 200]. По-умолчанию 20. Сервер может вернуть чуть больше, если сообщений в базе данных осталось мало.

**objectId** – id объекта (CAM:1, GRAY:5 и т.д). Если параметр не задан, то возвращаются события всех объектов.

Коды возврата:

200 - ОК

400 - неверный параметр (формат даты, например)

500 - ошибка

503 - ошибка соединения с ядром

504 - таймаут (ядро не вернуло данные в течение 2000 миллисекунд)

### Примеры ответа

#### XML:

```
<events>
```

```
  <event>
```

```
    <description>Запись выключена</description>
```

```
    <id>{E56B09A0-1A50-E211-840E-005056C00008}</id>
```

```
    <objectId>CAM:1</objectId>
```

```
    <ts>2012-12-27T15:43:27+04:00</ts>
```

```
  </event>
```

```
  <event>
```

```
    <description>Запись выключена</description>
```

```
    <id>{4482F63F-1A50-E211-840E-005056C00008}</id>
```

```
    <objectId>CAM:1</objectId>
```

```
    <ts>2012-12-27T15:40:50+04:00</ts>
```

```
  </event>
```



```
<event>
  <description>Запись выключена</description>
  <id>{35D4BE3E-1750-E211-840E-005056C00008}</id>
  <objectId>CAM:1</objectId>
  <ts>2012-12-27T15:19:16+04:00</ts>
</event>
</events>
```

#### JSON:

```
[ {
  "id" : "{E56B09A0-1A50-E211-840E-005056C00008}",
  "description" : "Запись выключена",
  "ts" : "2012-12-27T15:43:27.000+04:00",
  "objectId" : "CAM:1"
}, {
  "id" : "{4482F63F-1A50-E211-840E-005056C00008}",
  "description" : "Запись выключена",
  "ts" : "2012-12-27T15:40:50.000+04:00",
  "objectId" : "CAM:1"
}, {
  "id" : "{35D4BE3E-1750-E211-840E-005056C00008}",
  "description" : "Запись выключена",
  "ts" : "2012-12-27T15:19:16.000+04:00",
  "objectId" : "CAM:1"
} ]
```

## Отсылка команд на сервер

PUT

[http://example.com:\[port\]/web2/secure/configuration/GRAY:2/state/actions/DISARM/execute](http://example.com:[port]/web2/secure/configuration/GRAY:2/state/actions/DISARM/execute) - пример отсылки на сервер команды снятия с охраны Луча с идентификатором 2.

**Внимание!**

Название команды (в данном примере – DISARM) должно быть указано в верхнем регистре.

## Макрокоманды

**В разделе:**

- [Получение списка макрокоманд](#)
- [Получение параметров макрокоманд](#)
- [Запрос на выполнение макрокоманды на сервере](#)

Макрокоманды (макросы) – это некоторая предопределённая последовательность реакций на определённые события. Макрокоманды создаются на сервере и имеют ID и название. Они похожи на действия с объектами, но не привязаны к объекту.

### Получение списка макрокоманд

GET

`http://example.com:[port]/web2/secure/actions/`

**Пример ответа:**

```
<actions>
  <action>
    <description>Start recording by all cameras</description>
    <id>macro2</id>
  </action>
  <action>
    <description>Disarm all zones</description>
    <id>1</id>
  </action>
</actions>
```

### Получение параметров макрокоманд

Каких-либо дополнительных параметров у объекта нет. Можно ограничиться получением списка макросов.

GET

`http://example.com:[port]/web2/secure/actions/macro2/` - получение параметров макрокоманды с идентификатором macro2.

**Пример ответа:**

```
<action>
<description>Start recording by all cameras</description>
<id>macro2</id>
</action>
```

## Запрос на выполнение макрокоманды на сервере

PUT  
[http://example.com:\[port\]/web2/secure/actions/macro2/execute](http://example.com:[port]/web2/secure/actions/macro2/execute) – запрос на выполнение на сервере макрокоманды с идентификатором macro2.

## Видео

### Запрос миниатюр (скриншотов)

#### 1 способ

[http://example.com:\[port\]/web2/secure/video/image.jpg?cam.id=1&version=4.7.8.0](http://example.com:[port]/web2/secure/video/image.jpg?cam.id=1&version=4.7.8.0)

#### Параметры:

cam.id – обязательный. Id камеры.  
width -- необязательный. Значение может быть в диапазоне [64, 1600]. Сервер автоматически округляет ширину до большего значения, кратного 4.  
height – необязательный. Значение может быть в диапазоне [30, 1200].  
version – необязательный. версия клиента (на случай смены протокола). Сейчас нужно посылать значение "4.7.8.0".  
login – необязательный. Логин;  
password – необязательный. Если установлен доступ по паролю.

если width и height не установлены то размер возвращаемого изображения берётся из видеопотока.

#### Возвращаемое значение:

изображение jpeg приблизительно запрошенного размера.

Если произошла ошибка, то возвращается http код ошибки:

404 – камера отключена или не используется (disabled);  
403 – неверный пароль;  
426 – старая версия клиента;  
429 – слишком много запросов;  
444 – потерян сигнал по камере или камера отключена (коаксиальный провод отключен от платы);  
503 – ошибка архива.

#### Пример:

Получить изображение с камеры 5, шириной приблизительно 85 пикселей:  
<http://localhost:8079/web2/secure/video/image.jpg?cam.id=5&width=85&version=4.7.8.0&login=username&password=secret>

В ответ придёт картинка jpg шириной 88 пикселей, либо код об ошибке и body нулевой длины (т.е. придут только заголовки).

#### 2 способ

`http://example.com:[port]/web2/secure/video/action.do?version=4.9.0.0&command=frame.video&video_in=CAM:1&imageWidth=400&imageHeight=200`

cam.id – идентификатор камеры.

sessionId – любое значение.

imageWidth – ширина запрашиваемого кадра.

imageHeight – высота запрашиваемого кадра.

**Пример:**

Получить с Камеры 1 кадр высотой 400 пикселей.

`http://192.168.0.79:8085/web2/secure/video/action.do?version=4.9.0.0&command=frame.video&video_in=CAM:1&imageWidth=400`

## Запрос конфигурации

GET

`http://example.com:[port]/web2/secure/video/config.properties?version=4.7.8.0&login=XXX&password=YYY`

**Параметры:**

- version – обязательное поле. Версия клиента (на случай смены протокола). Сейчас нужно посылать значение "4.7.8.0".
- login – необязательное поле. Логин.
- password – необязательное поле. Используется, если установлен доступ по паролю.

**Особенности использования**

В начале работы неизвестно, установлены ли пароль, логин и т.п. Поэтому в первый раз необходимо послать следующий запрос:

GET

`http://www.examplehost.com/web2/secure/video/config.properties?version=4.7.8.0`

В ответ сервер отправит текстовый файл config.properties следующего формата:

password.enabled=true

login.enabled=true

password.invalid=true#



**Примечание.**

Символ # является признаком конца конфигурационного файла.

После получения файла такого вида можно понять, что пароль установлен и пароль неправильный. Неправильный он потому, что в данном случае был послан пустой пароль и пустой логин.

Необходимо запросить у пользователя логин и пароль и снова отослать серверу запрос на конфигурацию:

GET

`http://www.examplehost.com/web2/secure/video/config.properties?version=4.7.8.0&login=XXX&password=YYY`

Если пароль правильный или доступ разрешен без пароля, то сервер в ответ вышлет конфигурацию в следующем виде:

```
password.enabled=true
login.enabled=true
password.invalid=false
cam.0.id=2
cam.0.name=Face
cam.0.rights=11
cam.1.id=3
cam.1.name=Camera 3
cam.1.rights=11
cam.2.id=5
cam.2.name=Camera 5
cam.2.rights=11
cam.2.telemetry_id=1.1
cam.count=3#
```

`password.invalid=false` означает, что введён верный пароль.



**Примечание.**

Если разрешен доступ без пароля, то `password.enabled=false`, и вся нужная конфигурация будет получена с первого раза.

`cam.count=3` – общее количество камер в присланной конфигурации (`id` начинается с нуля).

Для каждой из трёх камер необходимо получить данные из конфигурации.

`cam.N.id` – `id` камеры.

`cam.N.name` – название камеры.

`cam.N.rights` – права.

`cam.N.telemetry_id` – `id` телеметрии (может отсутствовать, если телеметрии нет, тогда необходимо скрывать элементы управления телеметрией).

`cam.N.rights` – определяет права (они проверяются на сервере, но чтобы не показывать пользователю лишних опций, доступны и на клиенте). Параметр представляет собой флаги. Если флаг проставлен, то элемент интерфейса следует показывать, если нет, то скрывать.

```
static final int RIGHT_VIEW = 0x1; // доступен просмотр живого видео (этот всегда проставлен в 1)
```

```
static final int RIGHT_CONTROL = 0x2; // управление (телеметрия, постановка и снятие с охраны)
```

```
static final int RIGHT_CONFIG = 0x4; // reserved
```

```
static final int RIGHT_HISTORY = 0x8; // доступ к архиву
```

## Запрос видео

[http://example.com:\[port\]/web2/secure/video/action.do?version=4.7.8.0&sessionId=FC126734&cam.id=5&login=XXX&password=YYY](http://example.com:[port]/web2/secure/video/action.do?version=4.7.8.0&sessionId=FC126734&cam.id=5&login=XXX&password=YYY) - запрос видео для камеры с идентификатором 5.

cam.id – идентификатор камеры.

sessionId – любое значение.

Если указать в запросе версию 4.10.0.0, в результате будет получен поток в формате MJPEG без xml-вставок, который можно отображать на web-странице в браузерах Chrome и FireFox при помощи тэга IMG. Данная функция реализована как для живого, так и для архивного видео.



### Примечание.

Допускается одновременное получение не более 6 потоков видео.

Пример запроса:

```
http://10.0.36.158:8085/web2/secure/video/action.do?version=4.10.0.0&sessionId=1234567890&video_in=CAM:1&imageWidth=200&fps=1&login=1&password=1
```

Пример использования результатов запроса на web-странице:

```
<html>
<head/>
<body>
  
</body>
</html>
```

## Формат основного потока

В ответе приходит поток в виде:

```
HTTP/1.0 200 OK
Connection: close
Server: ITV-Intellect-Webserver/4.9.0.0
Cache-Control: no-store,no-cache,must-revalidate,max-age=0
Pragma: no-cache
Date: Mon, 13 Jan 2013 10:44:27 GMT
Content-Type: multipart/mixed;boundary=videoframe
```

```
--videoframe
Content-Type: text/xml
Content-Length: 138
```

```
<video_in>
  <sessionid>FC126734</sessionid>
  <video_in>CAM:5</video_in>
  <newstate>started</newstate>
  <errcode>100</errcode>
```

```
</video_in>
--videoframe
Content-Type: image/jpeg
Content-Length: 23978
X-Width: 320
X-Height: 240
X-Time: 2013-03-15T10:51:44.314+04:00
X-Timestamp: 0.000
```

```
<jpeg image>
--videoframe
Content-Type: image/jpeg
Content-Length: 23651
X-Width: 320
X-Height: 240
X-Time: 2013-03-15T10:51:44.314+04:00
X-Timestamp: 0.152
```

```
<jpeg image>
```

Здесь:

- X-Width - ширина изображения.
- X-Height- высота изображения.
- X-Time - абсолютное время формирования фрейма.
- X-Timestamp - относительное время фрейма в секундах (относительно начала потока).

В случае завершения потока по вине сервера может прийти завершающий пакет:

```
--videoframe
Content-Type: text/xml
Content-Length: 106
<video_in>
  <sessionid>FC126734</sessionid>
  <video_in>CAM:5</video_in>
  <newstate>closed</newstate>
  <errcode>103</errcode>
</video_in>
```

- sessionid - id сессии (тот же что и при старте).
- video\_in - идентификатор камеры.
- errcode - код ошибки:
  - 100 – отсутствие ошибки.
  - 101 – слишком много подключенных пользователей.
  - 102 – неверный пароль (пароль, теоретически, могут поменять в любой момент работы).
  - 103 – видео недоступно.
  - 104 – старая версия клиента. Обновите версию.

## Управление записью

### Начало записи

GET

[http://example.com:\[port\]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=1&target=CAM&targetid=1&command=REC&login=XXX&password=YYY](http://example.com:[port]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=1&target=CAM&targetid=1&command=REC&login=XXX&password=YYY)

### Окончание записи

GET

[http://example.com:\[port\]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=5&target=CAM&targetid=1&command=REC\\_STOP&login=XXX&password=YYY](http://example.com:[port]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=5&target=CAM&targetid=1&command=REC_STOP&login=XXX&password=YYY)

Здесь targetid==cam.id.

## Постановка и снятие с охраны камеры

### Постановка на охрану

GET

[http://example.com:\[port\]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=1&target=CAM&targetid=1&command=ARM&login=XXX&password=YYY](http://example.com:[port]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=1&target=CAM&targetid=1&command=ARM&login=XXX&password=YYY)

### Снятие с охраны

GET

[http://example.com:\[port\]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=5&target=CAM&targetid=1&command=DISARM&login=XXX&password=YYY](http://example.com:[port]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=5&target=CAM&targetid=1&command=DISARM&login=XXX&password=YYY)

Здесь targetid==cam.id.

## Управление телеметрией

GET



http://example.com:[port]/web2/secure/video/action.do?version=4.7.8.0&sessionid=29101F1&cam.id=5&target=PTZ&targetid=1.1&command=RIGHT&login=XXX&password=YYY&speed=2

### Параметры:

**command** – обязательный параметр. Может принимать следующие значения:

- RIGHT
- UP
- LEFT
- DOWN
- ZOOM\_IN
- ZOOM\_OUT
- GO\_PRESET – перейти в определенный пресет.
- POINTMOVE – зуммирование выделенной точки на изображении (x, y).
- AREA\_ZOOM – зуммирование выделенной области изображения (x,y,w,h).

**speed** – обязательный параметр. Скорость отработки команды (от 0 до 10). При управлении по сети из-за задержек лучше использовать низкие значения.

**cam.id** – обязательный параметр. Идентификатор камеры.

**target** – обязательный параметр. Всегда равно PTZ.

**targetid** – обязательный параметр. Id телеметрии, связанной с камерой (присылается в конфигурации SETUP).

**preset** – номер пресета (число). Обязательный параметр только для command=GO\_PRESET. В остальных случаях его значение игнорируется.

**x** – координата x относительно размера изображения. Может принимать значения от 0.0 до 1.0. Обязательный параметр только для command=POINTMOVE или command=AREA\_ZOOM. В остальных случаях его значение игнорируется.

**y** – координата y относительно размера изображения. Может принимать значения от 0.0 до 1.0. Обязательный параметр только для command=POINTMOVE или command=AREA\_ZOOM. В остальных случаях его значение игнорируется.

**w** – ширина области зуммирования y относительно размера изображения. Может принимать значения от 0.0 до 1.0. Обязательный параметр только для command=AREA\_ZOOM. В остальных случаях его значение игнорируется.

**h** – высота области зуммирования относительно размера изображения. Может принимать значения от 0.0 до 1.0. Обязательный параметр только для command=AREA\_ZOOM. В остальных случаях его значение игнорируется.

## Работа с архивом

### В разделе:

- Получение списка записей - "arc.intervals"
- Получение одного фрейма из архива - "arc.frame"
- Получение видео из архива - "arc.play"
- Получение списка записей (2-й способ)

Поток из видеоархива присылается в таком же формате, что и живое видео.

## Получение списка записей - "arc.intervals"

GET

`http://example.com:[port]/web2/secure/video/action.do?version=4.9.0.0&sessionId=29101F1&video_in=CAM:5&command=arc.intervals&time_from=2013-03-20T00:00:00.000+04:00&time_to=2013-03-22T23:59:59.999+04:00&max_count=100&split_threshold=10399&login=XXX&password=YYY`

Обязательные параметры:

`command=arc.intervals` – команда для получения списка записей

`video_in` – идентификатор камеры.

`time_from` – начало интересующего диапазона времени.

Необязательные параметры:

`time_to` – начало и конец интересующего диапазона времени.

`max_count` – максимальное количество записей в ответе

`split_threshold` – время для объединения нескольких интервалов (в секундах). Интервалы, расстояние между которыми будет меньше заданного, будут объединены в один.

В ответе придёт **XML**:

```
<?xml version="1.0" encoding="UTF-8"?>
<records>
  <record>
    <from>2011-09-01T00:00:00-05:00</from>
    <to>2011-09-01T00:00:35-05:00</to>
  </record>
  <record>
    <from>2011-09-01T00:00:35-05:00</from>
    <to>2011-09-01T00:01:10-05:00</to>
  </record>
</records>
```

## Получение одного фрейма из архива - "arc.frame"

GET

`http://example.com:[port]/web2/secure/video/action.do?version=4.9.0.0&sessionId=29101F1&video_in=CAM:5&command=arc.frame&time=2013-03-22T13:04:52.312+04:00&range=0.1&login=XXX&password=YYY`

Обязательные параметры:

`command=arc.frame` - команда для одного фрейма;

video\_in - идентификатор камеры;  
time - время, которое нас интересует.

Необязательные параметры:

range - время в секундах, для задания диапазона поиска ближайшего фрейма относительно time (если не указан, ищется ближайший по всему архиву);  
imageWidth - ширина в пикселях (если не указано или 0, рассчитывается автоматически с сохранением пропорций);  
imageHeight - высота в пикселях (если не указано или 0, рассчитывается автоматически с сохранением пропорций);  
fps - максимальная частота кадров в секунду (если не указано или 0, часта кадров не будет ограничиваться).

В ответ придут http-заголовки и ближайший фрейм из диапазона [time - range, time + range] в формате jpeg. Если фрейма в диапазоне не будет тело в ответе будет пустым.

#### **Получение видео из архива - "arc.play"**

GET

`http://example.com:[port]/web2/secure/video/action.do?version=4.9.0.0&sessionId=29101F1&video_in=CAM:5&command=arc.play&time_from=2013-03-22T13:04:52.312+04:00&time_to=2013-03-22T13:16:31.873+04:00&login=XXX&password=YYY`

Обязательные параметры:

command=arc.play - команда для получения видео из архива;  
video\_in - идентификатор камеры;  
time\_from - время начала проигрывания архива.

Необязательные параметры:

time\_to - время завершения проигрывания архива (если не указано, будет отдан весь архив до последней записи);  
imageWidth - ширина в пикселях (если не указано или 0, рассчитывается автоматически с сохранением пропорций);  
imageHeight - высота в пикселях (если не указано или 0, рассчитывается автоматически с сохранением пропорций);  
fps - максимальная частота кадров в секунду (если не указано или 0, часта кадров не будет ограничиваться).

При завершении потока придет завершающий пакет с newstate=closed и errcode=100.

#### **Получение списка записей (2-й способ)**

GET

http://example.com:[port]/web2/secure/archive/**CAM:2/[2011-12-30|2011-12]?[splitThreshold=50]&[days=1]**

splitThreshold – если разница между окончанием предыдущей записи и началом следующей меньше этого числа (в миллисекундах), то записи объединяются в одну. Чтобы никакие записи не объединялись, нужно указать splitThreshold=0. [default: 50].

days - количество дней, от текущего, за которые требуется получить архив. [default: 1]

Всё время интерпретируется как локальное для сервера.

Получить записи за 18 ноября 2013 года и слепить все записи, промежуток между которыми меньше 2000 миллисекунд:

http://example.com:[port]/web2/secure/archive/**CAM:1/2013-10-18/?splitThreshold=2000**

Получить записи за 10 дней, начиная с 18 ноября 2013 года:

http://example.com:[port]/web2/secure/archive/**CAM:1/2013-10-18/?days=10**

#### XML:

```
<?xml version="1.0" encoding="UTF-16"?>
<days>
  <day>
    <id>2013-11-10T00:00:00-02:00</id>
    <records>
      <from>2013-11-10T18:44:01.579-02:00</from>
      <to>2013-11-10T18:44:09.717-02:00</to>
    </records>
  </day>
  <day>
    <id>2013-11-18T00:00:00-02:00</id>
    <records>
      <from>2013-11-18T18:38:30.252-02:00</from>
      <to>2013-11-18T18:38:56.942-02:00</to>
    </records>
    <records>
      <from>2013-11-18T18:39:08.321-02:00</from>
      <to>2013-11-18T18:39:10.080-02:00</to>
    </records>
  </day>
</days>
```

#### JSON:

```
[ {
  "id" : "2013-11-10T00:00:00.000-02:00",
  "records" : [ {
    "from" : "2013-11-10T18:44:01.579-02:00",
    "to" : "2013-11-10T18:44:09.717-02:00"
  } ]
}, {
  "id" : "2013-11-18T00:00:00.000-02:00",
  "records" : [ {
    "from" : "2013-11-18T18:38:30.252-02:00",
    "to" : "2013-11-18T18:38:56.942-02:00"
  }, {
    "from" : "2013-11-18T18:39:08.321-02:00",
    "to" : "2013-11-18T18:39:10.080-02:00"
  } ]
} ]
```

Получение записей за месяц (показывает, в какие дни сентября есть записи):

[http://example.com:\[port\]/web2/secure/archive/CAM:2/2011-12/](http://example.com:[port]/web2/secure/archive/CAM:2/2011-12/)

**XML:**

```
<?xml version="1.0" encoding="UTF-8"?>
<days>
  <day>
    <id>2011-09-02T00:00:00-05:00</id>
  </day>
  <day>
    <id>2011-09-03T00:00:00-05:00</id>
  </day>
  <day>
    <id>2011-09-05T00:00:00-05:00</id>
  </day>
</days>
```

**JSON:**

```
[ {
  "id" : "2011-09-01T00:00:00-0500",
  "records" : [ ]
}, {
  "id" : "2011-09-03T00:00:00-0500",
  "records" : [ ]
}, {
  "id" : "2011-09-01T00:00:00-0500",
  "records" : [ ]
} ]
```

Если записей нет, то присылается

**XML:**

<days/>

**JSON:**

[]

## Нотификация

### В разделе:

- [Подписка на сообщения](#)
- [Аннулирование подписки](#)
- [Формат сообщения APN](#)

Используются системы нотификации APNS(iOS), C2DN (Android) и т.д.

deviceid – device token (APNs), registration id (в случае C2DN) и т.д.;

username – логин пользователя. Может быть пустой.

### Подписка на сообщения

Приложение при соединении с сервисом может осуществить подписку на сообщения APNS. В этом случае при выходе из программы на устройство будут приходить уведомления о тех или иных событиях.

POST

[http://example.com:\[port\]/web2/secure/subscription/](http://example.com:[port]/web2/secure/subscription/)

Ответ с кодом "201 Created" означает, что подписка прошла успешно.

Код 400 означает, что параметры заданы не верно (deviceId не должно быть пустым, должно быть длиной от 5 до 150 символов и содержать только цифры и буквы английского алфавита).

Тело POST должно содержать информацию о создаваемой подписке. Принимается только формат JSON. Требуется корректно проставлять заголовок Content-Type.

#### Пример ответа:

#### JSON

**Content-Type : application/json**

```
{
  "username" : "johndoe",
  "deviceid" : "somedeviceid"
}
```

#### Аннулирование подписки

Аннулирование подписки происходит в следующих случаях:

- Пользователь подписался на события с другого устройства;
- Сменился device token или registration id;
- Другой пользователь подписался на события с данного устройства;
- Произошла ручная отписка от сообщений.

DELETE

http://example.com:[port]/web2/secure/subscription/[deviceId]

Ответ с кодом "204 No Content" означает, что подписка прошла успешно.

#### Формат сообщения APN

```
{
  "aps" : {
    "alert" : "Motion Detected",
    "badge" : 2 //порядковый номер сообщения. Номера выдаются по порядку после момента последней подписки.
  },
  "e" : {
    "srv" : "XXX", //id сервера. Уникальный в рамках одного устройства iOS
    "stt" : 88, //id состояния (см. Список состояний для определённого класса объектов)
    "obj" : "6", //id объекта
    "ts" : "2010-08-02T23:30:00Z" //время отсылки события
  }
}
```

}

## Звук

### Получение живого звука

GET

http://example.com:[port]/web2/secure/video/action.do?version=4.9.0.0&sessionId=FC126734&command=audio.play&audio\_in=MIC:5&format=L16&login=XXX&password=YYY

sessionId – идентификатор сессии (тут пока не используется).

audio\_in – идентификатор аудиопотока.

format – формат аудиоданных (пока только L16).

В ответ будут получены аудиопакеты в следующем виде:

HTTP/1.0 200 OK

Connection: close

Server: ITV-Intellect-Webserver/4.9.0.0

Cache-Control: no-store,no-cache,must-revalidate,max-age=0

Pragma: no-cache

Date: Mon, 13 Jan 2013 10:44:27 GMT

Content-Type: multipart/mixed;boundary=audioframe

--audioframe

Content-Type: text/xml

Content-Length: 138

<audio\_in>

<sessionId>FC126734</sessionId>

<audio\_in>MIC:5</audio\_in>

<newstate>started</newstate>

<errcode>100</errcode>

</audio\_in>

--audioframe



Content-Type: audio/L16;rate=8000;channels=1

Content-Length: 1024

X-Time: 2013-03-22T13:16:31.371+04:00

<audio packet PCM16>

--audioframe

Content-Type: audio/L16;rate=8000;channels=1

Content-Length: 1278

X-Time: 2013-03-22T13:16:31.873+04:00

<audio packet PCM16>

Для остановки потока без разрыва соединения необходимо в этом же соединении отправить команду

GET

http://example.com:[port]/web2/secure/video/action.do?version=4.9.0.0&sessionId=29101F1&command=audio.stop&audio\_in=MIC:5&login=XXX&password=YYY

В этом случае в потоке придёт завершающий xml пакет:

--audioframe

Content-Type: text/xml

Content-Length: 106

<audio\_in>

<sessionId>FC126734</sessionId>

<audio\_in>MIC:5</audio\_in>

<newstate>closed</newstate>

<errcode>100</errcode>

</audio\_in>

## Проигрывание звука из архива

GET

http://example.com:[port]/web2/secure/video/action.do?version=4.9.0.0&sessionId=29101F1&command=arc.play&audio\_in=MIC:5&format=L16&time\_from=2013-03-22T13:16:31.873+04:00&time\_to=2013-03-22T13:04:52.312+04:00&login=XXX&password=YYY

audio\_in – идентификатор аудиопотока;

format – запрашиваемый формат (сейчас пока только L16);

time\_from - время начала проигрывания архива;

time\_to - время завершения проигрывания архива.

Поток приходит в том же виде, что и при живом звуке.

При завершении данных приходит завершающий xml пакет (как при получении живого звука – см. [Получение живого звука](#)).

## Отправка живого звука

Отправка звука идёт последовательной передачей пакетов командами:

POST

`http://example.com:[port]/web2/secure/video/action.do?version=4.9.0.0&sessionId=FC126734&command=audio.receive&audio_out=SPEAKER:3&login=XXX&password=YYY`

Content-type: audio/L16;rate=8000;channels=1

Connection: keep-alive

Далее происходит передача аудиопакета.

Формат звука – только L16.

rate – любое разумное значение.

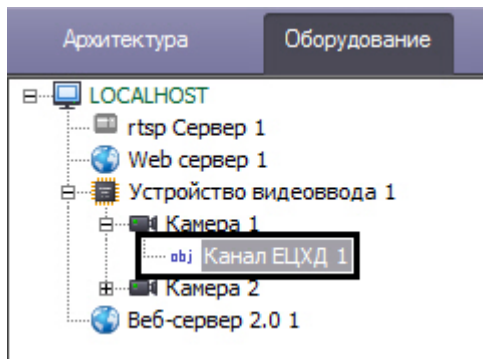
channels – от 1 до 6.

## Команды, используемые для интеграции ЕЦХД

ЕЦХД – государственная информационная система "Единый центр хранения и обработки данных".

В данном разделе описаны специальные http-запросы, используемые для интеграции ПК *Интеллект* с ЕЦХД. Для их работы необходимо, чтобы возможность использования таких запросов была включена на этапе настройки программы – см. [Выбор rtsp сервера для обработки запросов ЕЦХД](#).

Кроме того, запросы выполняются только для тех камер, на базе которых создан объект **Канал ЕЦХД**.



## Список камер и их параметры

В разделе:

- [GetCameras](#)
- [GetDeviceInfo](#)

## GetCameras

GET

http://example.com:[port]/getcameras

Возвращает список идентификаторов камер, заведенных на данном регистраторе/видеосервере. Дополнительно может содержать информацию о состоянии средства видеонаблюдения.

Пример:

GET http://192.168.15.182:8095/getcameras

Ответ:

```
{
"cameras": [
{
"id": 1,
"channel": 1,
"status": "working"
},
{
"id": 2,
"channel": 2,
"status": "signalLost"
}
]
}
```

## GetDeviceInfo

GET http://example.com:[port]/getdeviceinfo

Возвращает информацию об устройстве, такую как версия прошивки, название производителя, наименование модели и серийный номер.

Пример:

GET http://192.168.15.182:8095/getdeviceinfo

Ответ

```
{  
  "firmware version": "1.2.3 Rev B.",  
  "vendor": "Vendor Title Ltd"  
  "model": "Device Model",  
  "serial_number": "12345ABCDEF",  
  "ptz-status": "not supported"  
}
```

## Диапазоны доступных архивных записей

GET

http://example.com:[port]/getarchiveranges?cameraid=1

или

GET

http://example.com:[port]/getavailablearchiveranges?cameraid=1

Возвращает периоды времени, за которое доступны архивные записи с указанного средства видеонаблюдения.

**i** **Примечание.** По умолчанию фрагменты в ответе на запрос "склеиваются" (объединяются), если расстояние между ними составляет менее 5 секунд. Изменить данный период времени можно при помощи ключа реестра `SplitArchiveIntervals` (см. [Справочник ключей реестра](#)).

### Пример:

GET http://192.168.15.182:8095/getarchiveranges?cameraid=1

### Ответ:

```
{  
  "cameraid": 1,  
  "ranges": [  
    {  
      "from": 1412121600, //unixtime  
      "to": 1412172000  
    },  
    {  
      "from": 1412186400,  
      "to": 1412188200  
    }  
  ]  
}
```

## Работа с видеопотоками

### В разделе:

- [GetLiveUrl\(CameraId\)](#)
- [GetArcliveURL\(CameraId, FromDatetime, ToDatetime\)](#)

### GetLiveUrl(CameraId)

GET

`http://example.com:[port]/getliveurl?cameraid=1`

Возвращает rtsp URL «живого» видеопотока для указанной камеры.

Пример:

GET `http://192.168.15.182:8095/getliveurl?cameraid=1`

Ответ

```
{  
  "rtspurl": "rtsp://device-address/somelivemediastream0"  
}
```

### GetArcliveURL(CameraId, FromDatetime, ToDatetime)

GET

`http://example.com:[port]/getarchiveurl?cameraid=1&fromdatetime=2014-10-01T00:00:00&todatetime=2014-10-01T01:20:05`

Возвращает rtsp URL архивного видеопотока, отдаваемого регистратором, для указанной камеры, начиная с FromDateTime (и опционально, заканчивая EndDateTime).

Пример:

GET `http://192.168.15.182:8095/getarchiveurl?cameraid=1&fromdatetime=2014-10-01T00:00:00&todatetime=2014-10-01T01:20:05`

Ответ

```
{  
  "rtspurl": "rtsp://deviceaddress/somearchivemediastream?somedatetimetoken"  
}
```

## Выгрузка архивов

GET

http://example.com:[port]/downloadarchivefile?camera id =1&fromdatetime=2014-10-01T00:00:00&todatetime=2014-10-01T01:20:05

**Пример:**

GET http://192.168.15.182:8095/downloadarchivefile?camera id =1&fromdatetime=2014-10-01T00:00:00&todatetime=2014-10-01T01:20:05

Ответ

HTTP/1.1 200 OK

Content-Type: application/octet-stream

## Управление функциями средства видеонаблюдения

### В разделе:

- [Дискретное движение](#)
- [Относительное движение](#)
- [Установка положения средства видеонаблюдения](#)
- [Получение положения средства видеонаблюдения](#)
- [Фокусировка](#)
- [Управление диафрагмой](#)
- [Ночной режим](#)
- [Подсветка](#)
- [Черно-белый режим](#)

### Дискретное движение

https://example.com:[port]/?cameraID={1}& ip={2}& login={3}&pass={4}& action={5}&x={6}&y={7}&z = {8}&modelName={9}

Атомарный сдвиг средства видеонаблюдения в указанном направлении.

cameraID - идентификатор средства видеонаблюдения.

ip - IP-адрес средства видеонаблюдения.

login - учетная запись средства видеонаблюдения.

pass - пароль доступа к средству видеонаблюдения.

action - имя команды [*degreesmove*].

x - поворот в плоскости PAN [-180 ..0.. 180].

y - поворот в плоскости TILT [-180 ..0.. 180].

z - увеличение/уменьшение зума [0.. 100].

modelName - модель средства видеонаблюдения.

### Относительное движение

https://example.com:[port]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9}

Поворот средства видеонаблюдения относительно текущего положения. Область видимости средства видеонаблюдения делится на сетку, где центральная точка имеет координаты (x:0, y:0), левая верхняя (x:-7, y:7), правая нижняя (x:7, y:-7). Поворот средства видеонаблюдения должен быть осуществлен таким образом, чтобы объект по указанным в команде координатам оказался в центре изображения средства видеонаблюдения.

Допускается «оптическая» погрешность, возникающая в результате расстояния до объекта видимости.

Погрешность, возникающую за счет проекции сферы на плоскость, следует компенсировать.

*cameraID* - идентификатор средства видеонаблюдения.

*ip* - IP-адрес средства видеонаблюдения.

*login* - учетная запись средства видеонаблюдения.

*pass* - пароль доступа к средства видеонаблюдения.

*action* - имя команды (*degreestove2*).

*x* - поворот в плоскости PAN [-7..0..7].

*y* - поворот в плоскости TILT [-7..0..7].

*Z* - увеличение/уменьшение зума [-1..0.. 1].

*modelName* - модель средства видеонаблюдения.

### **Установка положения средства видеонаблюдения**

`https://example.com:[port]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9}`

Перевод средства видеонаблюдения в указанное положение в градусах относительно «0» позиции.

*cameraID* - идентификатор средства видеонаблюдения.

*ip* - IP-адрес средства видеонаблюдения.

*login* - учетная запись средства видеонаблюдения.

*pass* - пароль доступа к средству видеонаблюдения.

*action* - имя команды (*setposition*).

*x* - установка в плоскости PAN [-180 ..0.. 180].

*y* - установка в плоскости TILT [-180 ..0..180].

*Z* - значение зума [0.. 100].

*modelName* - модель средства видеонаблюдения.

### **Получение положения средства видеонаблюдения**

`https://example.com:[port]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&modelName={6}`

Получение положение средства видеонаблюдения в плоскостях PAN и

TILT в градусах, а также текущие значение зума.

**Ответ** в формате JSON:

```
{"y":56, "x":105, "z":0}
```

*cameraID* - идентификатор средства видеонаблюдения.

*ip* - IP-адрес средства видеонаблюдения.

*login* - учетная запись средства видеонаблюдения.

*pass* - пароль доступа к средству видеонаблюдения.

*action* - имя команды (*getposition*).

*modelName* - модель средства видеонаблюдения.

### Фокусировка

```
https://example.com:[port]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9}
```

Команда фокусировки средства видеонаблюдения, где параметр *z* управляет поведением фокуса:

- 1: Увеличить фокус
- -1 : Уменьшить фокус
- 0: Авто

*cameraID* - идентификатор средства видеонаблюдения.

*ip* - IP-адрес средства видеонаблюдения.

*login* - учетная запись средства видеонаблюдения.

*pass* - пароль доступа к средству видеонаблюдения.

*action* - имя команды (*focus*).

*x* - не используется [0].

*y* - не используется [0].

*z* - управление фокусом [-1..0..1].

*modelName* - модель средства видеонаблюдения.

### Управление диафрагмой

```
https://example.com:[port]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9}
```

Команда управления диафрагмой средства видеонаблюдения, где параметр *z* управляет поведением диафрагмы:

- 1: Открыть диафрагму
- -1 : Закрыть диафрагму
- 0: Авто



*cameraID* - идентификатор средства видеонаблюдения.

*ip* - IP-адрес средства видеонаблюдения.

*login* - учетная запись средства видеонаблюдения.

*pass* - пароль доступа к средству видеонаблюдения.

*action* - имя команды (*iris*).

*x* - не используется [0].

*y* - не используется [0].

*z* - управление диафрагмой [-1..0..1].

*modelName* - модель средства видеонаблюдения.

### Ночной режим

[https://example.com:\[port\]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9}](https://example.com:[port]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9})

Доступны следующие режимы работы средства видеонаблюдения:

- 1: Дневной режим
- -1 : Ночной режим

*cameraID* - идентификатор средства видеонаблюдения.

*ip* - IP-адрес средства видеонаблюдения.

*login* - учетная запись средства видеонаблюдения.

*pass* - пароль доступа к средству видеонаблюдения.

*action* - имя команды (*switch\_day\_night*).

*x* - не используется [0].

*y* - не используется [0].

*z* - управление режимом [-1..0..1].

*modelName* - модель средства видеонаблюдения.

### Подсветка

[https://example.com:\[port\]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9}](https://example.com:[port]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9})

Доступны следующие режимы работы подсветки:

- 1: Включить
- -1: Выключить

*cameraID* - идентификатор средства видеонаблюдения.

*ip* - IP-адрес средства видеонаблюдения.

*login* - учетная запись средства видеонаблюдения.

*pass* - пароль доступа к средству видеонаблюдения.

*action* - имя команды (*backlight*).

*x* - не используется [0].

*y* - не используется [0].

*z* - управление режимом [-1..0..1].

*modelName* - модель средства видеонаблюдения.

### Черно-белый режим

https://example.com:[port]/?cameraID={1}&ip={2}&login={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9}

Доступны следующие режимы работы средства видеонаблюдения:

- 1: Включить
- -1: Выключить

*cameraID* - идентификатор средства видеонаблюдения.

*ip* - IP-адрес средства видеонаблюдения.

*login* - учетная запись средства видеонаблюдения.

*pass* - пароль доступа к средству видеонаблюдения.

*action* - имя команды (*switch\_color*).

*x* - не используется [0].

*y* - не используется [0].

*z* - управление режимом [-1..0..1].

*modelName* - модель средства видеонаблюдения.

## Экспорт архива

### В разделе:

- [Создание задания на экспорт архива](#)
- [Получение статуса экспорта](#)
- [Удаление архива](#)

По умолчанию экспорт архива осуществляется в формат *mp4*. Изменить формат можно при помощи ключа реестра *ExportContainerFormat* (см. [Справочник ключей реестра](#)).

### Создание задания на экспорт архива

Пример:

POST http://192.168.15.182/createarchivetask  
Content Type: application/json  
Content:

```
{  
  "CameraId": "1",  
  "From": "2016-06-27T15:10:00.00Z",  
  "To": "2016-06-27T15:20:00.00Z"  
}
```

Ответ:

```
{  
  "CameraId" : "1",  
  "From" : "2016-06-27T15:10:00.00Z",  
  "To" : "2016-06-27T15:20:00.00Z",  
  "ArchiveTaskId" : "084b56a5-bd49-4327-82db-9bc911f7ff96",  
  "ErrorMessage" : null,  
  "State" : "Created"  
}
```

В папке экспорта (по умолчанию C:\Users\User\Documents\Intellect\export) создаётся папка с соответствующим именем (084b56a5-bd49-4327-82db-9bc911f7ff96) и mp4-файлом внутри.

### Получение статуса экспорта

Пример:

GET http://192.168.15.182/getarchivetaskstatus?archivetaskid=104b38d4-07d7-4d2f-84da-49b3e255d2bf

Ответ:

```
{  
  "Percents" : 100,  
  "Url" : "http://192.168.15.182/download?file=104b38d4-07d7-4d2f-84da-49b3e255d2bf",  
  "CameraId" : "1",  
  "From" : "2016-06-27T15:10:00.000+03:00",  
  "To" : "2016-06-27T15:11:00.000+03:00",  
  "ArchiveTaskId" : "104b38d4-07d7-4d2f-84da-49b3e255d2bf",  
  "ErrorMessage" : "null",  
  "State" : "ReadyForDownload"  
}
```

По ссылке в строке "URL" можно скачать искомый mp4-файл.

### Удаление архива

Пример:

DELETE http://192.168.15.182/removearchive?archivetaskid=084b56a5-bd49-4327-82db-9bc911f7ff96

Ответ:

```
{
  "ArchiveTaskId" : "084b56a5-bd49-4327-82db-9bc911f7ff96",
  "ErrorMessage" : null,
  "Success" : true
}
```

Из папки экспорта удалится соответствующая папка с mp4-файлом.

## Отправка реакций и событий в ПК Интеллект по HTTP-запросу

ПК *Интеллект* принимает на порт 10112 команды и события вида

`http://[IP Сервера ПК Интеллект]:10112/intellect_core/React?command="[команда в формате ПК Интеллект]"`

`http://[IP Сервера ПК Интеллект]:10112/intellect_core/Event?command="[событие в формате ПК Интеллект]"`



### Примечание.

Указанный порт можно изменить при помощи ключа реестра RestPort, подробнее данный ключ описан в [Справочнике ключей реестра](#).

### Примеры:

Добавление субтитров на видеоизображение с камеры 2 при помощи HTTP-запроса:

`http://localhost:10112/intellect_core/React?command="CAM|2|ADD_SUBTITLES|command<Some text\n!>"`

Сгенерировать тревогу по камере 2 при помощи HTTP-запроса:

`http://localhost:10112/intellect_core/Event?command="CAM|2|MD_START"`

При получении команд описанного вида в ПК *Интеллект* будут генерироваться обычные события и реакции, которые можно по необходимости использовать в скриптах и макрокомандах (см. [Руководство Администратора](#), раздел [Создание и использование макрокоманд](#), а также [Руководство по программированию \(JScript\)](#)).

## Заключение

Более подробная информация о программном комплексе *Интеллект* содержится в следующих документах:

1. [Руководство администратора](#);
2. [Руководство оператора](#);
3. [Руководство по установке и настройке компонентов охранной системы](#);
4. [Руководство по программированию](#);
5. [Руководство по программированию \(JScript\)](#).

Если в процессе работы с данным программным продуктом у вас возникли трудности или проблемы, вы можете связаться с нами. Однако рекомендуем предварительно сформулировать ответы на следующие вопросы:

1. В чем именно заключается проблема?
2. Когда и после чего появилась данная проблема?
3. В каких именно условиях проявляется проблема?

Помните, что чем более полную и подробную информацию вы нам предоставите, тем быстрее наши специалисты смогут устранить вашу проблему.

Мы всегда работаем над улучшением качества своей продукции, поэтому будем рады любым вашим предложениям и замечаниям, касающимся работы нашего программного обеспечения, а также документации к нему.

Пожелания и замечания по данному Руководству следует направлять в Отдел технического документирования компании Ай-Ти-Ви групп ([documentation@itv.ru](mailto:documentation@itv.ru)).

## ПРИЛОЖЕНИЕ 1. Описание структуры ddi-файла

Описание полей таблицы, расположенной на вкладке **Имена** (раздел **<Objects>**), приведено в таблице ниже.

Поле	Описание
Имя (<ObjectName>)	Идентификационное имя объекта
Название (<VisibleName>)	Отображаемое имя
Имя группы (<GroupName >)	Имя группы объектов. Используется для объединения объектов в группу в дереве настроек ПК <i>Интеллект</i>

Описание полей таблицы, расположенной на вкладке **События** (раздел **<Events>**), приведено в таблице ниже.

Поле	Описание
Название (<EventName>)	Идентификационное имя события.
Описание (<EventDescription>)	Описание сообщения, выводимое в протоколе событий.
Обработка сообщений (<EventType>)	Определение цвета фона сообщения в протоколе событий: обычное – без цвета; тревожное – красное окно; информационное – синее окно.
Звуковая поддержка (<IsSoundEnabled>)	Воспроизведение звукового файла при срабатывании сообщения.
Не слать в сеть (<IsNetworkDisabled>)	Не посылать сообщение по сети.
Не протоколировать (<IsProtocolDisabled>)	Не отображать сообщение в протоколе событий.
Журнал Windows (<IsWindowsLogEnabled>)	Сохранять сообщения в журнале событий Windows. <i>Примечание. Запись в журнал Windows невозможна, если событие не протоколируется</i>

Описание полей таблицы, расположенной на вкладке **Реакции** (раздел **<Reacts>**), приведено в таблице ниже.

Поле	Описание
Название (<ReactName>)	Имя реакции
Описание (<ReactDescription>)	Описание реакции, выводимое в контекстном меню при щелчке правой кнопкой мыши по значку объекта на <i>Карте</i>
Флаги (<IsReactArm>)	Флаг выполнения реакции: либо для одного объекта, либо для группы объектов, входящих в один раздел

Описание полей таблицы, расположенной на вкладке **Значки** (раздел **<Icons>**), приведено в таблице ниже.

Поле	Описание
Имя файла (<FileName>)	Часть имени bmp-файла, которая является идентификатором изображения. Идентификатор изображения позволяет использовать несколько bmp-файлов для представления на <i>Карте</i> объектов одного типа (см. раздел <a href="#">Использование утилиты ddi.exe для работы с DDI-файлами</a> )
Название (<IconName>)	Описание bmp-файла объекта

Описание полей таблицы, расположенной на вкладке **Состояния** (раздел **<States>**), приведено в таблице ниже.

Поле	Описание
Название (<StateName>)	Имя состояния
Изображение (<ImgName>)	Часть имени, которая является идентификатором состояния, соответствующего bmp-файла (см. раздел <a href="#">Использование утилиты ddi.exe для работы с DDI-файлами</a> ).  <i>Примечание. Объекты на Карте могут быть отображены с помощью линий, т.е. без использования bmp-файлов. В этом случае, если изменяется состояние объекта, меняется цвет линии. Цвет (RGB) состоянию задается следующим образом: &lt;Состояние&gt;\$R:G:B</i>
Описание (<StateDescription>)	Описание состояния
Мерцание (<IsStateFlashing>)	Отображение на <i>Карте</i> : обычное – отсутствие мерцания, тревожное – мерцание значка на <i>Карте</i>

Описание полей таблицы, расположенной на вкладке **Правила перехода** (раздел **<Rules>**), приведено в таблице ниже.

Поле	Описание
Событие (<EventName>)	Событие, по которому выполняется переход
Переход из состояния (<FromStateName>)	Исходное состояние, из которого должен быть осуществлен переход
Переход в состояние (<ToStateName>)	Итоговое состояние, в которое должен быть осуществлен переход

## ПРИЛОЖЕНИЕ 2. Объявление классов NissObjectDLLExt и CoreInterface

### На странице:

- [CoreInterface](#)
- [NissObjectDLLExt](#)

## CoreInterface

```
class CoreInterface
{
public:
    virtual BOOL DoReact (React&) = 0;
    virtual BOOL NotifyEvent(Event&) = 0;
    virtual void SetupACDevice(LPCTSTR objtype, LPCTSTR objid, LPCTSTR objtype_reader) = 0;

    virtual  BOOL  IsObjectExist(LPCTSTR objtype, LPCTSTR id) = 0;
    virtual  BOOL  IsObjectDisabled(LPCTSTR objtype, LPCTSTR id) = 0;
    virtual Msg FindPersonInfoByCard(LPCTSTR facility_code, LPCTSTR card) = 0;
    virtual Msg FindPersonInfoByExtID(LPCTSTR external_id) = 0;
    virtual  CString GetObjectName (LPCTSTR objtype, LPCTSTR id) = 0;
    virtual  CString GetObjectState(LPCTSTR objtype, LPCTSTR id) = 0;
    virtual  void    SetObjectState(LPCTSTR objtype, LPCTSTR id, LPCTSTR state) = 0;
    virtual  BOOL  IsObjectState(LPCTSTR objtype, LPCTSTR id, CString state) = 0;

    virtual  CString GetObjectParam (LPCTSTR objtype, LPCTSTR id, LPCTSTR param) = 0;
    virtual  int  GetObjectParamInt (LPCTSTR objtype, LPCTSTR id, LPCTSTR param) = 0;
    virtual  CMapStringToStringArray* GetObjectParamList(LPCTSTR objtype, LPCTSTR id, LPCTSTR param) = 0;
    virtual  CStringArray* GetObjectParamList(LPCTSTR objtype, LPCTSTR id, LPCTSTR param, LPCTSTR name) = 0;
```

```
virtual void    GetObjectParams (LPCTSTR objtype, LPCTSTR id, Msg& msg) = 0;  
virtual void    SetObjectParamInt (LPCTSTR objtype, LPCTSTR id, LPCTSTR param, int val) = 0;  
virtual CString GetObjectIdByParam(LPCTSTR type, LPCTSTR param, LPCTSTR val) = 0;  
virtual CString GetObjectIdByName(LPCTSTR type, LPCTSTR name) = 0;  
virtual CString GetObjectParentId(LPCTSTR objtype, LPCTSTR id, LPCTSTR parent) = 0;  
virtual int GetObjectIds(LPCTSTR objtype, CStringArray& list, LPCTSTR main_id = NULL) = 0;
```



```
virtual int GetObjectChildIds(LPCTSTR objtype, LPCTSTR objid, LPCTSTR childtype, CStringArray& list) = 0;
};
```

## NissObjectDLLExt

```
class NissObjectDLLExt
{
protected:
    CoreInterface* m_pCore;
public:
    NissObjectDLLExt(CoreInterface* core) { m_pCore = core; }

    virtual CString GetObjectType() = 0;
    virtual CString GetParentType() = 0;
    virtual int GetPos() { return -1; }
    virtual CString GetPort() { return CString(); }
    virtual CString GetProcessName() { return CString(); }
    virtual CString GetDeviceType() { return CString(); }

    virtual BOOL HasChild() { return FALSE; }
    virtual UINT HasSetupPanel() { return FALSE; }
    virtual void OnPanelInit(CWnd*) {}
};
```

```
virtual void OnPanelLoad(CWnd*,Msg&) {}  
virtual void OnPanelSave(CWnd*,Msg&) {}  
virtual void OnPanelExit(CWnd*) {}  
virtual void OnPanelButtonPressed(CWnd*,UINT) {}  
virtual BOOL IsRegionObject() { return FALSE; }  
virtual BOOL IsProcessObject() { return FALSE; }  
virtual BOOL IsIncludeParentId()          { return 0; }  
virtual BOOL IsWantAllEvents()            { return 0; }  
virtual CString DescribeSubscribeObjectsList() { return CString(); }  
virtual CString GetIncludeIdParentType(){ return CString(); }  
virtual CString DescribeParamLists(){ return CString(); }  
  
virtual void OnCreate(Msg&) {}  
virtual void OnChange(Msg&,Msg&) {}  
virtual void OnDelete(Msg&) {}  
virtual void OnInit(Msg&)      {}  
virtual void OnEnable(Msg&) {}  
virtual void OnDisable(Msg&) {}  
virtual BOOL OnEvent(Event&) { return FALSE; }
```

```
virtual BOOL OnReact(React&) { return FALSE; }  
};
```