



Руководство по интеграции аппаратно-программных модулей

1. Руководство по интеграции аппаратно-программных модулей. Введение.	4
2. Интеграция аппаратно-программных модулей с ПК Интеллект	4
2.1 Общие сведения об интеграции аппаратно-программных модулей	4
2.2 Редактирование DBI-файла	5
2.2.1 Добавление объектов в intellect.dbi	5
2.2.2 Использование утилиты ddi.exe для работы с DBI-файлами	8
2.3 Редактирование DDI-файла	10
2.3.1 Добавление в intellect.ddi информации об объекте	10
2.3.2 Использование утилиты ddi.exe для работы с DDI-файлами	13
2.4 Дополнительные возможности утилиты ddi.exe	17
2.5 Разработка MDL-файла	18
2.5.1 Мастер создания MDL-файла	27
2.6 Разработка RUN-файла	28
2.7 Создание и настройка интегрированных объектов (модулей) в ПК Интеллект	29
3. INTELLECT INTEGRATION DEVELOPER KIT (IIDK)	32
3.1 Общие сведения об IIDK	32
3.1.1 Назначение IIDK	32
3.1.2 Требования к разработчику	33
3.1.3 Состав IIDK	33
3.2 Подключение к ПК Интеллект	33
3.2.1 Параметры подключения	33
3.2.2 Объект Интерфейс IIDK	34
3.2.3 Настройка пересылки событий через объект Интерфейс IIDK	35
3.3 Функции IIDK	36
3.3.1 Connect	36
3.3.2 SendMsg	37
3.3.3 Disconnect	38
3.3.4 Другие функции	39
3.4 Синтаксис отправляемых сообщений	42
3.4.1 Синтаксис сообщений	42
3.4.2 Синтаксис сообщений (900 порт)	43
3.4.3 Использование классов Event и React	44
3.5 Примеры управления объектами системы	44
3.5.1 Добавление, изменение и удаление объектов системы	45
3.5.2 Особенности работы с системой в многопользовательском режиме	45
3.5.3 Определение компьютера, на котором был выгружен ПК Интеллект (через 1030 порт)	46
3.5.4 Вывод видеочамеры на монитор	46
3.5.5 Получение параметров объекта (через 1030 порт). GET_CONFIG	46
3.5.6 Получение информации о состоянии объекта. GET_STATE и GET_LIST	46

3.5.7 Вывод информационного сообщения. SET_STATE	47
3.5.8 Работа с живым и архивным видео	47
3.5.9 Управление телеметрией	48
3.5.10 Операции со слоем карты	48
4. Заключение	48
5. ПРИЛОЖЕНИЕ 1. Описание структуры ddi-файла	49
6. ПРИЛОЖЕНИЕ 2. Объявление классов NissObjectDLLExt и CoreInterface	50

Руководство по интеграции аппаратно-программных модулей.

Введение.

Документ [Руководство по интеграции аппаратно-программных модулей](#) содержит сведения, необходимые для внедрения в систему функциональных модулей, обеспечивающих решение следующих задач:

1. Добавление нового охранного оборудования в систему.
2. 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 аппаратно-программных модулей](#).

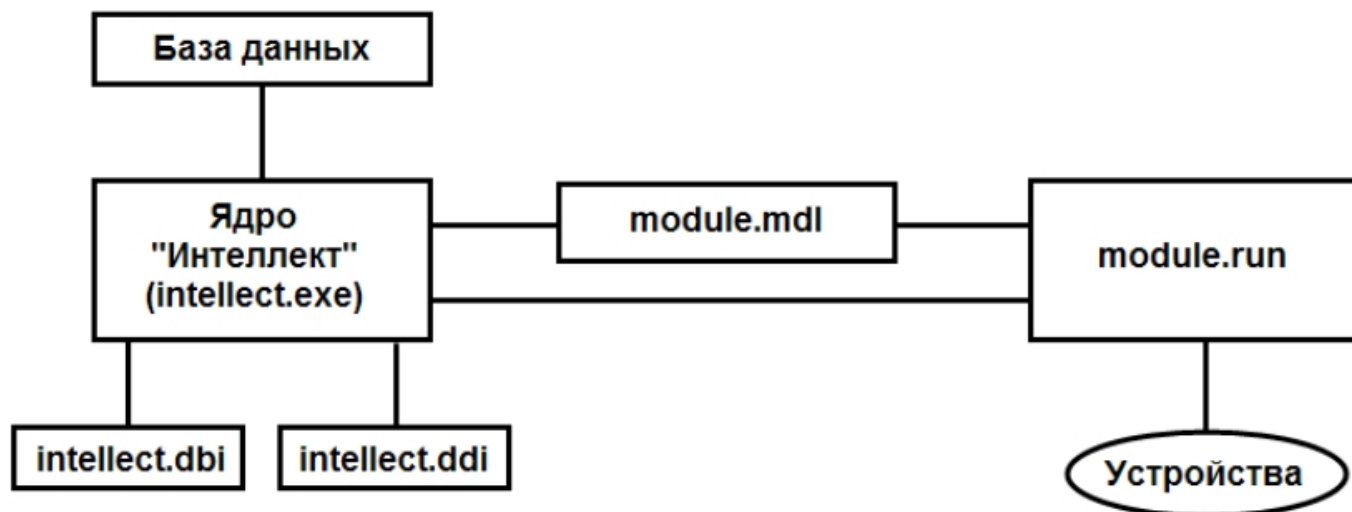
Интеграция аппаратно-программных модулей с ПК Интеллект

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 исполнительному 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 – гggг-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

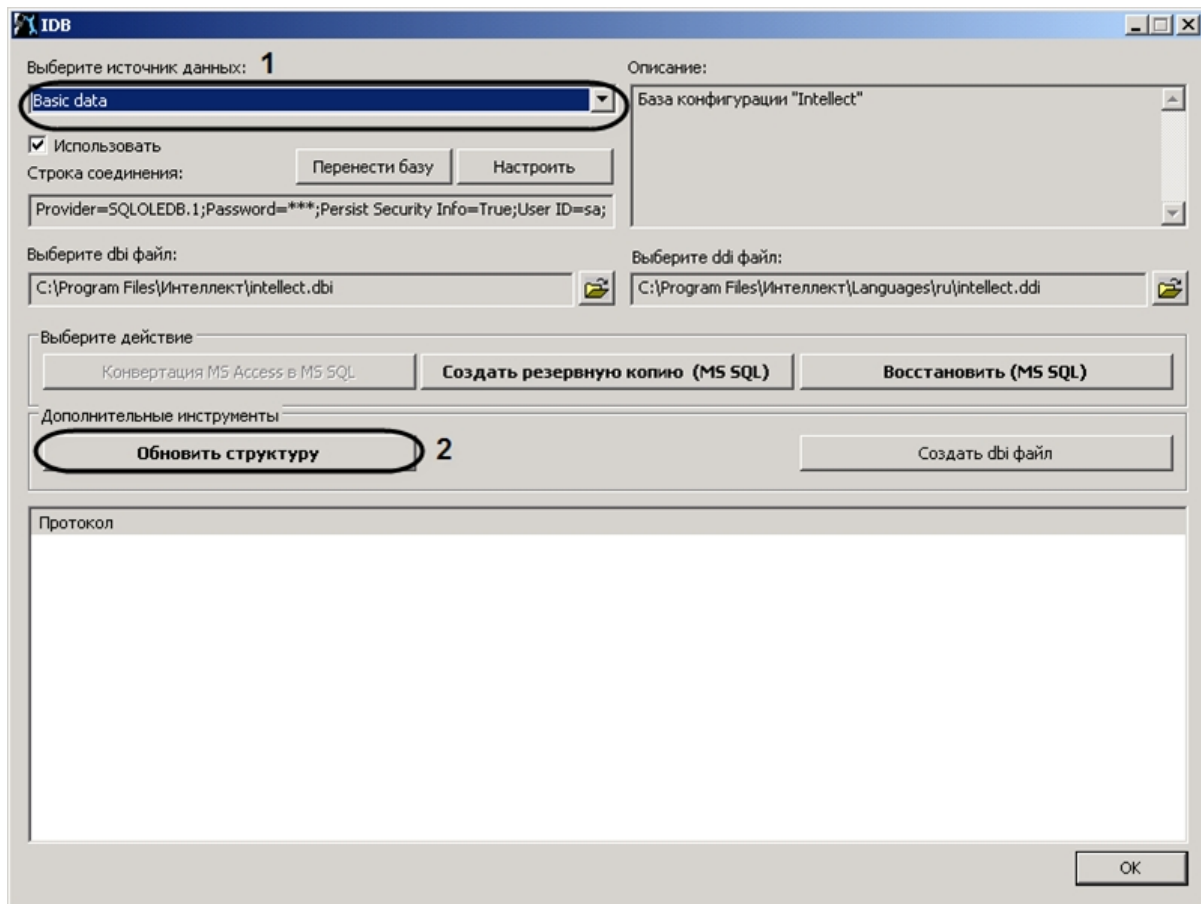
x, INTEGER
y, INTEGER
w, INTEGER
h, INTEGER
exe, CHAR, 255
guid, UNIQUEIDENTIFIER

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

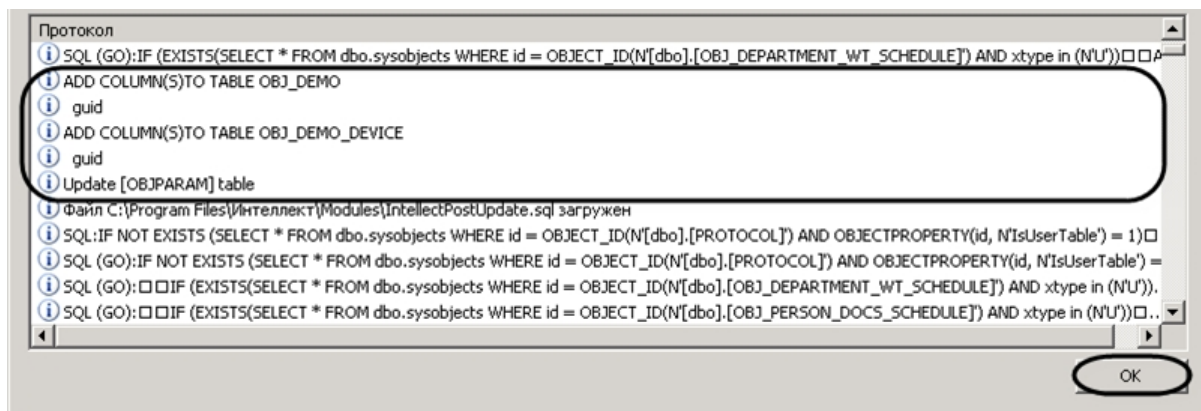
[OBJ_DEMO]
id, CHAR, 16
name, CHAR, 60
parent_id, CHAR, 16
port, CHAR, 5
flags, INTEGER

[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:

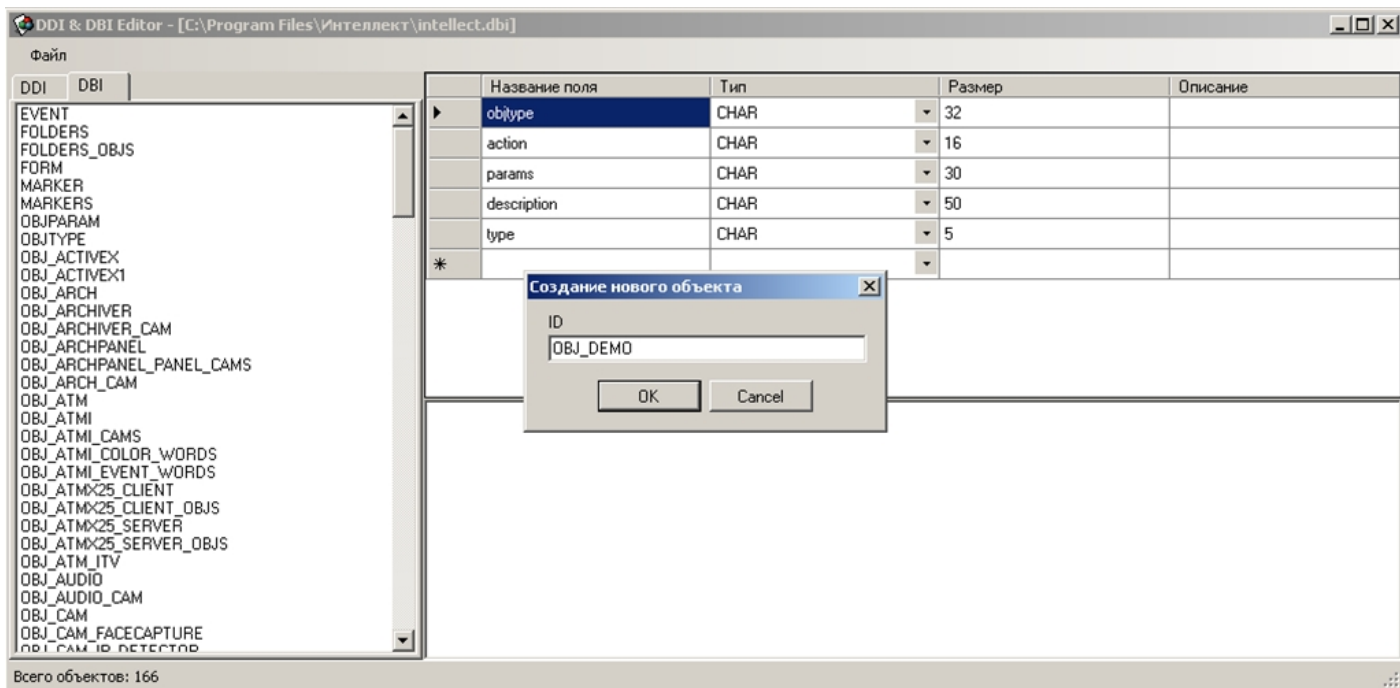
1. Запустить утилиту *ddi.exe*, расположенную в каталоге *Интеллект\Tools*.
2. В окне утилиты перейти на вкладку **DBI**.
3. В меню **Файл** выбрать пункт **Открыть**. В результате выполнения операции появится диалоговое окно **Открыть**.
4. Выбрать файл *intellect.dbi*, расположенный в корне директории 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**.



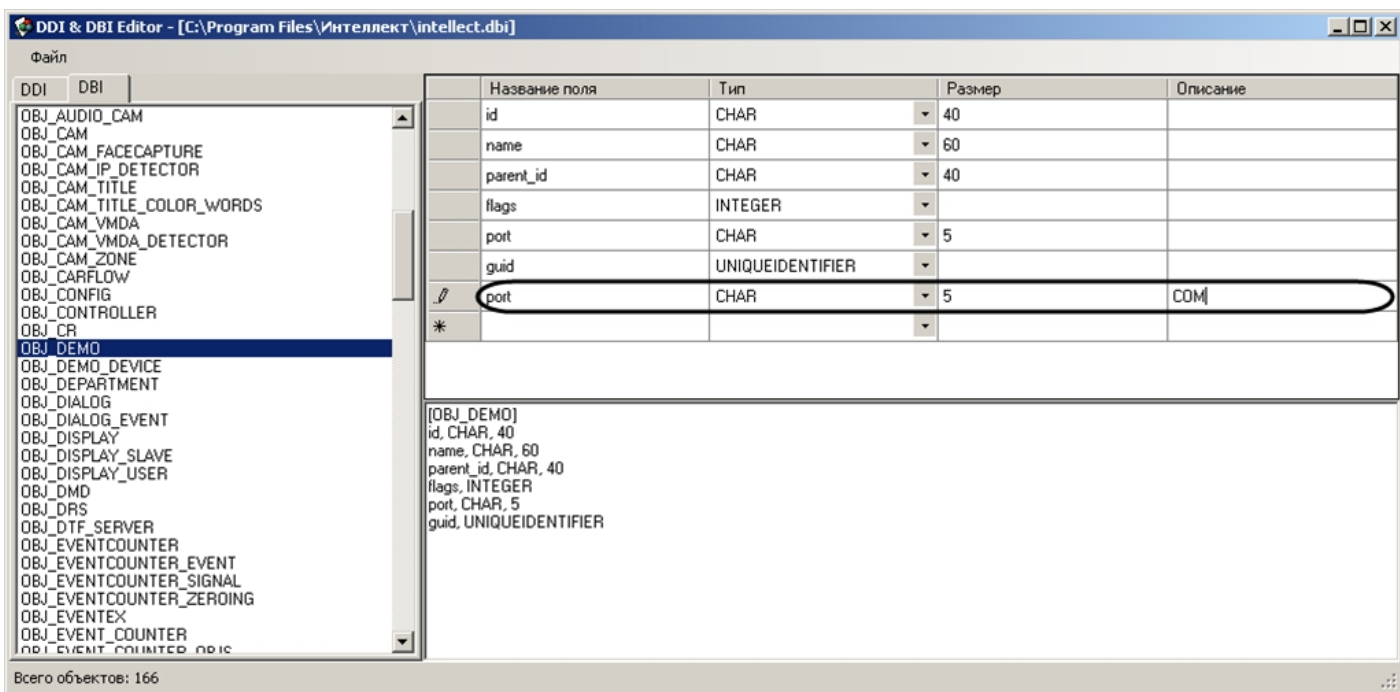
Примечание.

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.

Добавление в `intellect.ddi` информации об объекте

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

Для добавления информации об объекте **DEMO** 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!

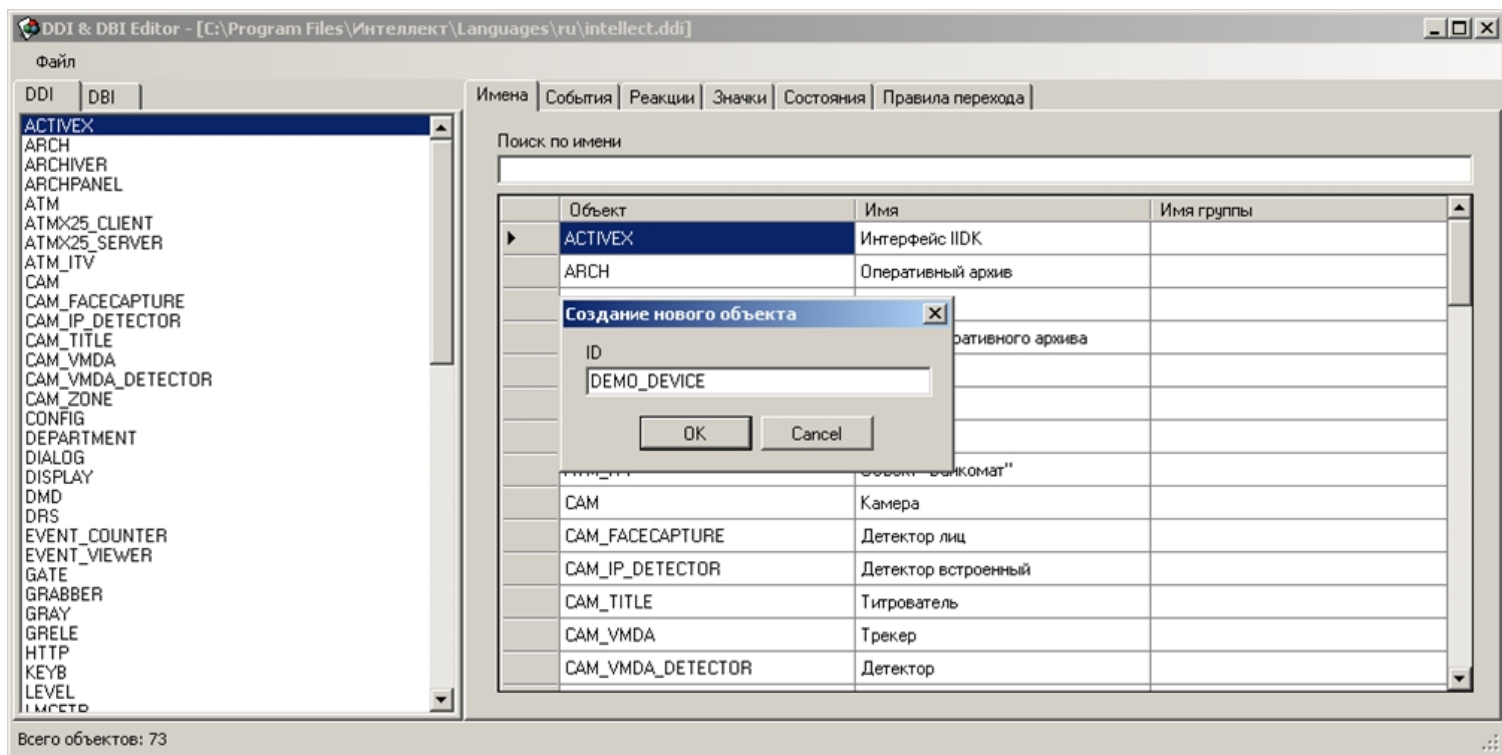
После изменения DDI-файлов требуется обновить структуру базы данных с помощью утилиты `idb.exe` (см. шаги 4-7 раздела [Добавление объектов в intellect.dbi](#)).

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`.

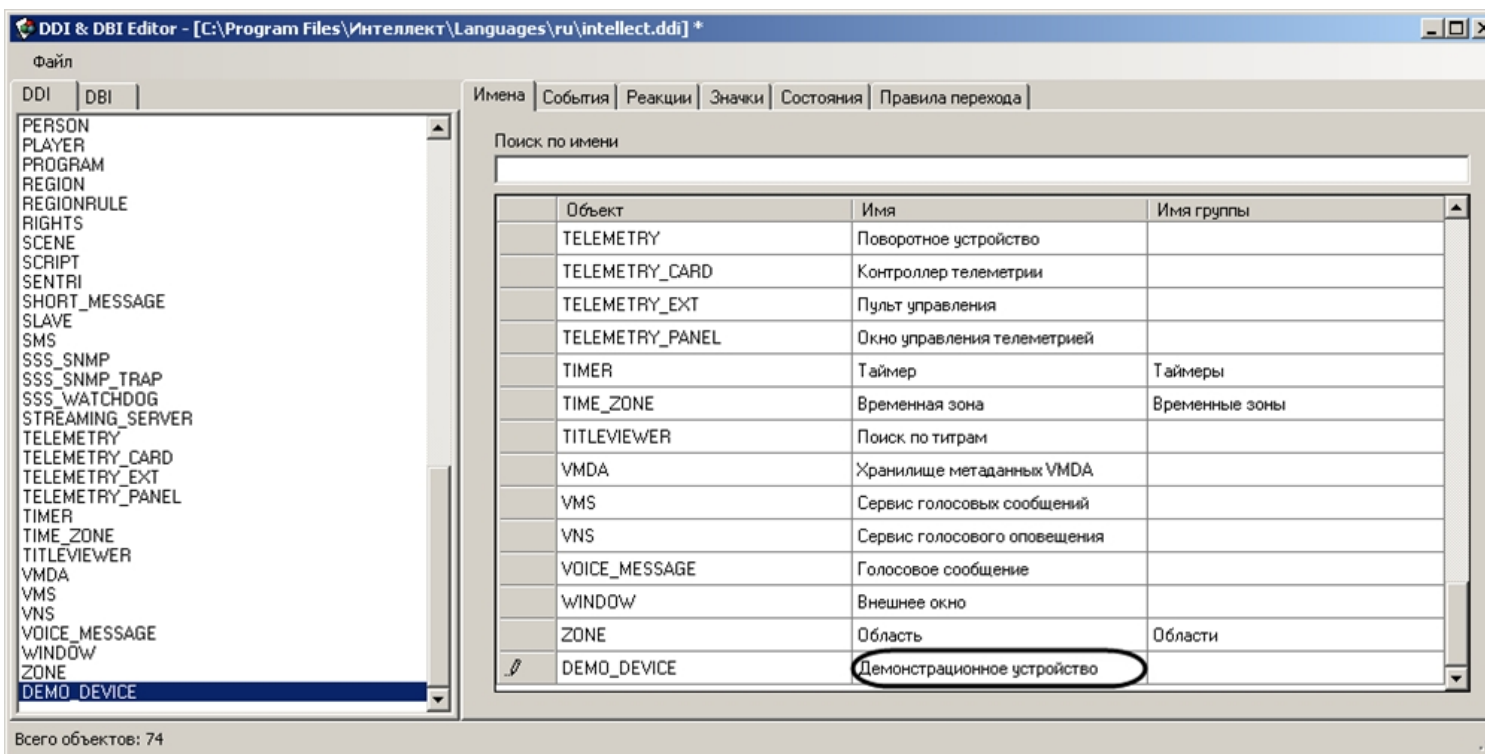
Для добавления в `intellect.ddi` информации об объекте **DEMO_DEVICE** необходимо выполнить следующие действия:

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](#)

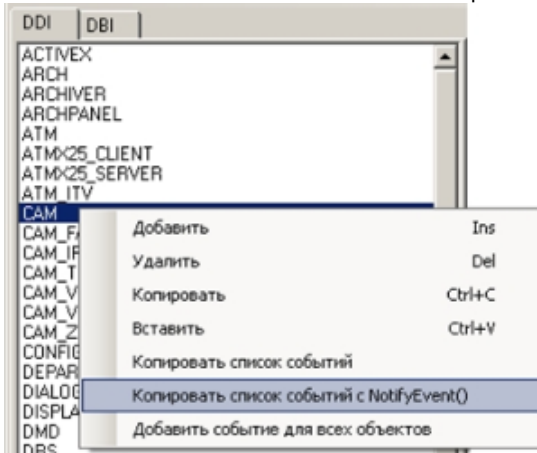
Внимание!

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

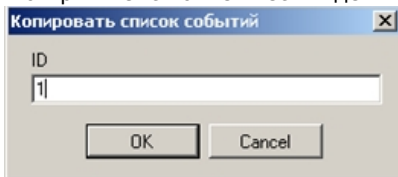
Дополнительные возможности утилиты ddi.exe

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:

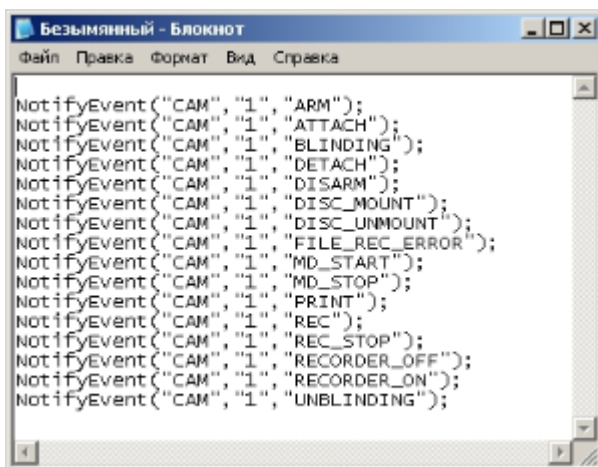
1. В контекстном меню списка объектов выбрать пункт **Копировать список событий с NotifyEvent()** .



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

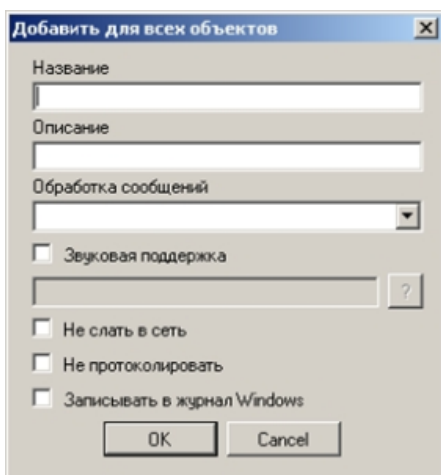


Копирование списка событий завершено. Буфер обмена будет содержать события объекта в виде, представленном на рисунке.



```
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". Внимание! Этот параметр должен быть согласован с компанией «Ай Ти Ви групп»	<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>Внимание! Этот параметр должен быть согласован с компанией «Ай Ти Ви групп»</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: Пользователи, Временная зона и Уровни доступа</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 Точка прохода</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& params) { CString s; s = arams.GetParam("port"); pwnd->GetDlgItem(IDC_PORT)-> SetWindowText(s); }</pre>

<p>virtual void OnPanelSave(CWnd*,Msg&)</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& params) { CString s; pwnd-> GetDlgItem(IDC_PORT)-> 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 1151. Например, для кнопки Test идентификатор в файле <i>Resource.h</i> определяется как:</i></p> <p>#define IDC_TEST 1151</p>	<pre>Virtual void OnPanelButtonPressed (CWnd* pwnd,UINT id) { if(id==IDC_TEST) { React react("DEMO",Id,"TEST"); m_pCore->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&)</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& msg) { msg.SetParam ("port","COM1"); }</pre>
<pre>virtual void OnInit(Msg&)</pre>	<p>Используется при инициализации объекта. В качестве параметра передается ссылка на сообщение, содержащее информацию по объекту</p>	<pre>virtual void OnInit (Msg& msg) { OnChange (msg, msg); }</pre>
<pre>virtual void OnChange(Msg&,Msg&)</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& msg, Msg& prev) { React react (msg.GetSourceType(), msg.GetSourceId(),"INIT"); react.SetParam("port",msg.GetParam("port")); m_pCore->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& msg) { React react (msg.GetSourceType(), msg.GetSourceId(),"EXIT"); m_pCore-> DoReact(react); }</pre>
virtual void OnEnable(Msg&)	Предназначен для обработки нажатия кнопки Disable 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 Disable 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& 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 mdl-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)

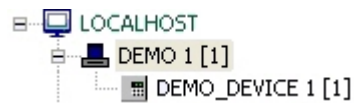
Используя функцию WinAPI – *PostThreadMessage*, необходимо перехватить это сообщение и обеспечить корректную выгрузку модуля. В VC++ и MFC сообщение **WM_EXIT** отлавливается в классе, наследуемом от *CWinApp*, в Delphi и CBuilder – *TApplication*.

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:

4. Произвести настройку объектов.

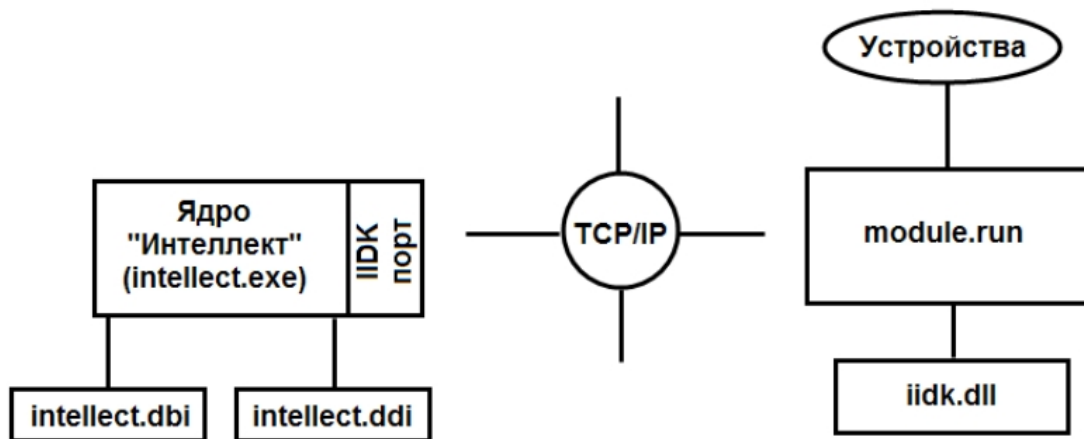
Процесс создания и настройки интегрированных объектов в ПК *Интеллект* завершен.

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.

- b. Для объекта **Интерфейс IIDK** – порт 1030.
- c. Для банкоматов – порт 1009.
- 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.

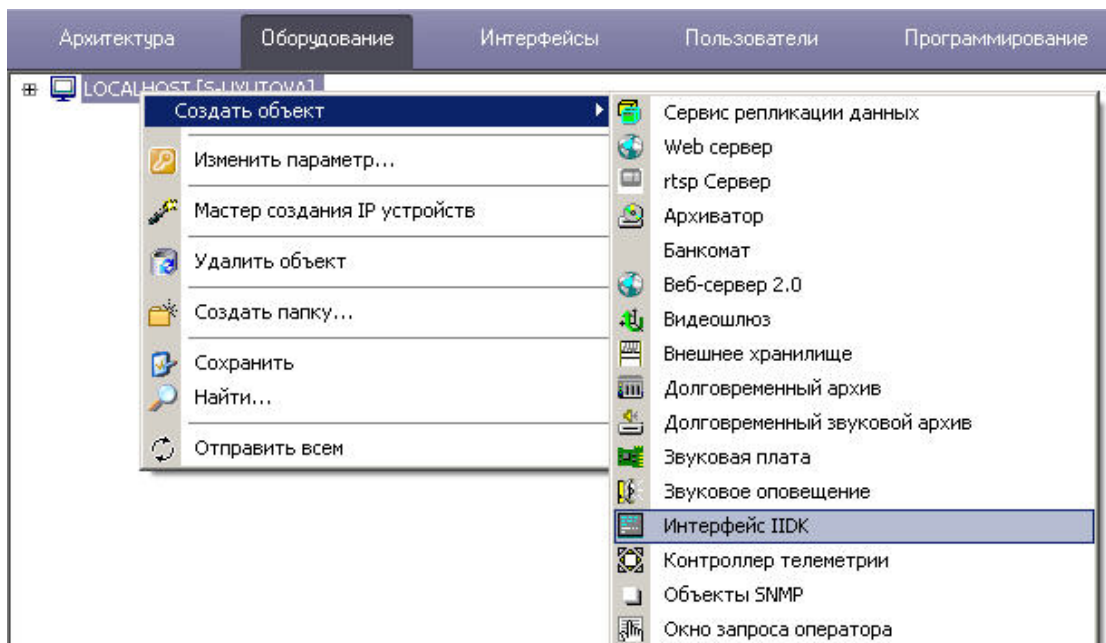
 **Примечание.**
Если создан mdl-файл (см. раздел [Разработка MDL-файла](#)), для подключения к ядру ПК *Интеллект* объект **Интерфейс IIDK** в системе не создается. В качестве идентификатора подключения передается пустая строка, то есть id равен "".

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.

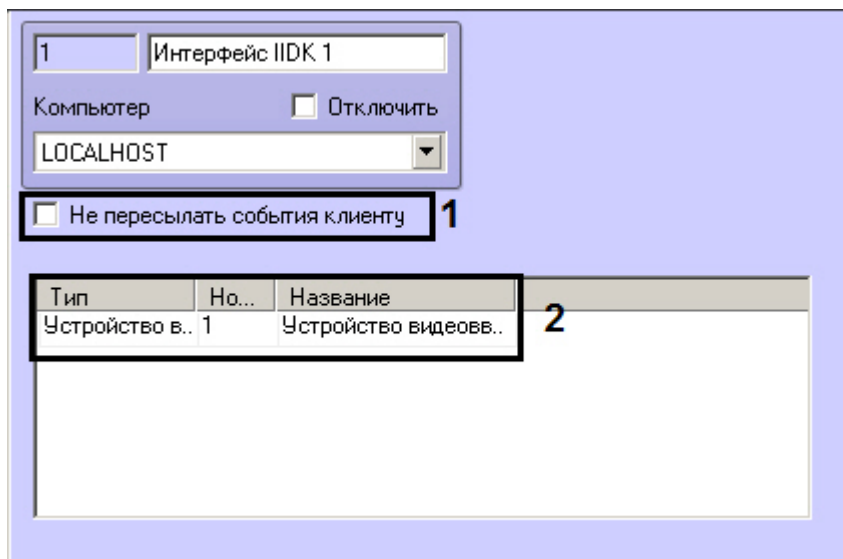
При использовании распределенной архитектуры ПК *Интеллект* объект **Интерфейс IIDK** должен быть создан на компьютере, содержащем программное ядро, к которому выполняется подключение. В случае, если подключение выполняется к компьютеру, на котором установлено *Рабочее место мониторинга*, то в параметрах 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.

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";</pre>
LPCTSTR port	порт TCP/IP, через которое происходит подключение	
LPCTSTR id	идентификатор подключения, для видео	

<pre>_stdcall *func)(LPCTSTR msg))</pre>	Callback-функция, принимающая сообщения от ПК <i>Интеллект</i>	<pre> BOOL IsConnect = Connect(ip, port, id, myfunc); if (!IsConnect) { // не удалось подключиться AfxMessageBox("Error"); } </pre>
--	--	---

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

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)
```

Параметр	Описание
<code>ip</code>	IP-адрес Сервера ПК <i>Интеллект</i> ,
<code>port</code>	Порт TCP/IP, через которое происходит подключение
<code>id</code>	Идентификатор подключения <i>slave</i> , для видео
<code>lpfunc</code>	Callback-функция, принимающая сообщения от ПК <i>Интеллект</i>
<code>user_param</code>	Дополнительный параметр, который будет приходить в Callback-функцию, чтобы разделить слайвы, если функция одна на всех.
<code>async_connect</code>	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.
<code>connect_attempts</code>	Количество попыток подключения

SendReactToCore

Функция предназначена для отправки реакции в указанное ядро.

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

Параметр	Описание
id	Идентификатор подключения ядра
msg	Отправляемое сообщение. Формат сообщения аналогичен SendMsg .

IsConnected

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

```
BOOL IsConnected(LPCTSTR id);
```

Параметр	Описание
id	Идентификатор подключения ядра

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 React
<pre>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);</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!**

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.

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.

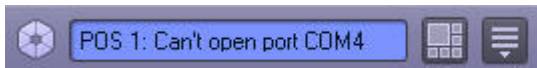
Вывод информационного сообщения. SET_STATE

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<MAPLAYER>,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 **MAPLAYER|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<MAPLAYER>,source_id<1>,action<ACTIVATE>**
2. Посылкой реакции на порт 1030: **MAPLAYER|1|ACTIVATE**

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).

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 Использование утилиты ddi.exe для работы с DDI-файлами)
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 Использование утилиты ddi.exe для работы с DDI-файлами). <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: <Состояние>\$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; }  
};
```