Руководство по программированию (JScript)

1. Программирование с использованием языка JScript 5
1.1 Назначение и возможности языка JScript 5
1.2 Описание объектной модели языка JScript в программном комплексе Интеллект 5
1.2.1 Объект Core и его встроенные методы 5
1.2.1.1 Объект Core 5
1.2.1.2 Метод SetObjectParam 5
1.2.1.3 Метод SetObjectState 6
1.2.1.4 Метод DebugLogString 7
1.2.1.5 Метод Base64Decode 7
1.2.1.6 Метод Sleep 8
1.2.1.7 Метод Itv_var 9
1.2.1.8 Метод Int_var 10
1.2.1.9 Метод GetObjectParentType 11
1.2.1.10 Метод GetIPAddress 12
1.2.1.11 Метод GetObjectName 13
1.2.1.12 Метод GetObjectState 14
1.2.1.13 Метод GetObjectParam 15
1.2.1.14 Метод GetObjectParentId 15
1.2.1.15 Метод DoReactStr 16
1.2.1.16 Метод DoReact 18
1.2.1.17 Метод DoReactSetupCore 19
1.2.1.18 Метод DoReactSetup 20
1.2.1.19 Метод DoReactGlobal 21
1.2.1.20 Метод NotifyEventStr 21
1.2.1.21 Метод NotifyEvent 23
1.2.1.22 Метод NotifyEventGlobal 24
1.2.1.23 Метод CreateMsg 24
1.2.1.24 Методы Lock и Unlock 26
1.2.1.25 Метод IsAvailableObject 28
1.2.1.26 Метод GetUserId 29
1.2.1.27 Метод GetEventDescription 29
1.2.1.28 Метод GetObjectIdByParam 30
1.2.1.29 Метод SaveToFile 31
1.2.1.30 Метод GetLinkedObjects 32
1.2.1.31 Метод WriteIni 33
1.2.1.32 Метод ReadIni 33
1.2.1.33 Метод AddIni 34
1.2.2 Объекты MsgObject и Event и их встроенные методы и свойства 34
1.2.2.1 Объекты MsgObject и Event 34

1.2.2.2 Метод GetSourceType . . . 35
1.2.2.3 Метод GetSourceId . . . 36
1.2.2.4 Метод GetAction . . . 37
1.2.2.5 Метод GetParam . . . 37
1.2.2.6 Метод SetParam . . . 38
1.2.2.7 Метод MsgToString . . . 38
1.2.2.8 Метод StringToMsg . . . 39
1.2.2.9 Метод StringToParams . . . 40
1.2.2.10 Метод Clone . . . 41
1.2.2.11 Метод GetObjectIds . . . 42
1.2.2.12 Метод GetObjectParams . . . 43
1.2.2.13 Свойство SourceType . . . 43
1.2.2.14 Свойство SourceId . . . 44
1.2.2.15 Свойство Action . . . 44
1.3 Инструментарий программирования на JScript . . . 45
1.3.1 Системный объект Скрипт . . . 45
1.3.2 Утилита Редактор-Отладчик . . . 47
1.3.3 Отладочное окно . . . 48
1.3.3.1 Включение Отладочного окна . . . 48
1.3.3.2 Работа с Отладочным окном . . . 50
1.3.3.2.1 Копирование информации о событии или реакции в буфер обмена . . . 51
1.3.3.2.2 Выделение сообщений цветом . . . 52
1.3.3.2.3 Фильтр событий и реакций . . . 54
1.3.3.2.4 Поиск событий и реакций . . . 56
1.3.3.2.5 Очистка Отладочного окна . . . 57
1.3.4 Получение списка системных названий объектов, реакций и событий ПК Интеллект . . . 57
1.4 Создание, сохранение и удаление скрипта . . . 60
1.4.1 Создание скрипта . . . 60
1.4.2 Сохранение скрипта . . . 63
1.4.3 Удаление скрипта . . . 63
1.5 Создание первого скрипта . . . 63
1.6 Отладка скриптов . . . 68
1.6.1 Возможности отладки скриптов . . . 68
1.6.2 Создание и использование тестовых событий . . . 69
1.6.3 Работа с отладочными окнами утилиты Редактор-Отладчик . . . 71
1.6.3.1 Типы отладочных окон: Script Messages и Информация от потока . . . 71
1.6.3.2 Отображение сообщений о запуске, проверке, изменении и выполнении скриптов в отладочных окнах . . . 73
1.6.4 Использование сторонних программ-отладчиков . . . 75
1.7 Примеры скриптов на языке JScript . . . 77

2. Заключение 80
3. Приложение 1. Описание утилиты Редактор-Отладчик 80
3.1 Назначение утилиты Редактор-Отладчик 80
3.2 Описание интерфейса утилиты Редактор-Отладчик 80
3.2.1 Интерфейс утилиты Редактор-Отладчик 80
3.2.2 Вкладка Script Debug/Edit 81
3.2.3 Вкладка Script Messages 83
3.2.4 Главное меню 86
3.2.4.1 Описание интерфейса главного меню 86
3.2.4.2 Описание пункта главного меню Файл 86
3.2.4.3 Описание пункта главного меню Вид 87
3.2.4.4 Описание пункта главного меню Отладка и редактирование 87
3.2.4.5 Описание элементов пункта главного меню Список сообщений 87
3.2.5 Описание диалогового окна Фильтр 88
3.2.6 Описание диалогового окна Выделить цветом 89
3.2.7 Описание панели инструментов утилиты Редактор-Отладчик 90
4. Приложение 2. Создание виртуальных объектов с возможностью задавать события, реакции и состояния 92
4.1 Назначение виртуальных объектов и их реализация в ПК Интеллект 93
4.2 Пример создания виртуального объекта 93
4.2.1 Подготовка файла dbi 93
4.2.2 Подготовка файла ddi 94
4.2.3 Подготовка файла xml 97
4.2.4 Создание и использование виртуального объекта в ПК Интеллект 98

Программирование с использованием языка JScript

Назначение и возможности языка JScript

В составе программного комплекса *Интеллект* язык программирования JScript позволяет реализовывать дополнительные пользовательские функции, не предусмотренные основными функциональными возможностями программы.

Язык программирования JScript является стандартным средством разработки пользовательских скриптов. В программном комплексе *Интеллект* поддерживается версия языка JScript, реализованная в технологии ActiveX корпорации Microsoft. Описание объектной модели языка JScript, используемого в программном комплексе *Интеллект,* приведено в документации корпорации Microsoft (например, MSDN).

Интерпретация скриптов, разработанных на языке JScript для программного комплекса *Интеллект,* выполняется с использованием стандартных (входящих в поставку ОС Windows) программных компонентов ActiveX. Поэтому при разработке скриптов допускается использование любых компонентов объектной модели версии языка JScript, реализованной в технологии ActiveX.

Программный комплекс *Интеллект* дополнительно предоставляет специализированную объектную модель для разработки скриптов на языке JScript, позволяющую работать с системными объектами программного комплекса *Интеллект*, получать и отправлять системные события и реакции.

Описание объектной модели языка JScript в программном комплексе Интеллект

Объект Core и его встроенные методы

Объект Core

Объект **Core** – это глобальный статический объект, реализующий методы, используемые для контроля состояния и управления системными объектами программного комплекса *Интеллект*. Методы объекта **Core** позволяют получать сведения о зарегистрированных системных объектах, генерировать для них реакции, изменять состояния. Объект **Core** р еализует дополнительные методы для приостановки выполнения скриптов, их отладки, создания глобальных переменных и обращения к ним.

Объект **Core** не является прототипом, и создание других объектов на его основании (т.е. с использованием объекта **Core** как шаблона) не допускается. Все методы объекта **Co re** являются статическими. Таким образом, вызов методов объекта **Core** осуществляется непосредственно из скрипта без обращения к самому объекту **Core**.

Метод SetObjectParam

Метод SetObjectParam используется для задания значений параметрам системных объектов.

Синтаксис обращения к методу:

```
function  SetObjectParam(objtype: String,  id: String,  param : String, value : String)
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, параметрам которого требуется задать значения. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **param** – обязательный аргумент. Соответствует параметру системного объекта. Допустимые значения: тип String, диапазон ограничен допустимыми для заданного объекта параметрами.
4. **value** – обязательный аргумент. Соответствует значению, задаваемому параметру param системного объекта. Допустимые значения: тип String, диапазон зависит от

устанавливаемого параметра.

Пример. По запуску Макрокоманды № 1 проверять, настроены ли камеры №№ 1 – 4 на передачу цветного видеосигнала. При обнаружении камеры, настроенной на передачу черно-белого видеосигнала переводить ее в режим работы в цвете (устанавливая ее параметру **Цветность** (“color”) значение true (“1”)).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")

{

 var i;
 for(i=1;i<=4;i=i+1)
 {
  if (GetObjectParam("CAM",i,"color") == "0")
  {
   SetObjectParam("CAM",i,"color","1");
  }
 }
}
```

Примечание.
В случае, если на момент запуска скрипта активен изменяемый в нем объект (т.е. открыта панель его настроек), то изменение параметров объекта методом SetObjectParam не будет произведено. Например, если открыта панель настроек объекта **Камера 1** и запущен вышеприведенный скрипт, режим работы камеры №1 не будет изменен на цветной.

Метод SetObjectState

Метод SetObjectState используется для изменения состояний системных объектов.

Синтаксис обращения к методу:

```
function  SetObjectState(objtype : String, id : String, state : String)
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, состояние которого требуется изменить. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом objtype типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **state** – обязательный аргумент. Соответствует состоянию, в которое требуется перевести объект. Допустимые значения: тип String, диапазон ограничен допустимыми для заданного объекта состояниями.

Пример. Каждый час проверять поставлена ли камера №1 на охрану. В том случае, если камера №1 снята с охраны, поставить ее на охрану.

Примечание.
Предварительно необходимо создать объект **Таймер** с идентификационным номером 1. Установить параметру **Минуты** объекта **Таймер** значение 30. В данном случае таймер будет срабатывать каждый час, например, следующим образом: в 09:30, 10:30, 11:30 и т.д.

```
if (Event.SourceType == "TIMER" && Event.SourceId == "1" && Event.Action == "TRIGGER")
{
 if (GetObjectState("CAM", "1") == "DISARMED")
 {
  SetObjectState("CAM", "1", "ARMED");
 }
}
```

Метод DebugLogString

Метод DebugLogString используется для вывода пользовательских сообщений в отладочные окна утилиты *Редактор - Отладчик*.

Синтаксис обращения к методу:

```
function  DebugLogString(output : String)
```

Аргументы метода:

1. **output** – обязательный аргумент. Задает строку сообщения, которую требуется вывести в отладочное окно утилиты *Редактор – Отладчик*. Допустимые значения: тип String.

Пример. При регистрации в системе какого-либо события от любого из микрофонов выводить его в отладочное окно.

```
if (Event.SourceType == "OLXA_LINE")
{
 var msgstr = Event.MsgToString();
 DebugLogString("Событие от микрофона  " + msgstr);
}
```

Метод Base64Decode

Метод Base64Decode используется для декодирования строк, закодированных по схеме Base64.

Синтаксис обращения к методу:

```
function Base64Decode(data_in: String, WideChar: Boolean)
```

Аргументы метода:

1. **data_in** – обязательный аргумент. Задает строку в Base64, которую необходимо декодировать;
2. **WideChar** – обязательный аргумент. Определяет тип кодировки. Возможные значения 0 или 1. Если тип кодировки Unicode, то значение аргумента —1, иначе 0.

Пример. По запуску макрокоманды №1 декодировать строку, заданную в Base64. Вывести результат декодирования в отладочное окно утилиты *Редактор - Отладчик*. (Результатом является строка «Intellect JScript»)

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")

{
 var str = Base64Decode("SW50ZWxsZWN0IEpTY3JpcHQ= ", 0);
 DebugLogString(str);
}
```

Метод Sleep

Метод Sleep используется для приостановки выполнения скрипта на заданное время.

Синтаксис обращения к методу:

```
function  Sleep(milliseconds : int)
```

Аргументы метода:

1. **milliseconds** – обязательный аргумент. Задает время, на которое требуется приостановить выполнение скрипта. Указывается в миллисекундах. Допустимые значения: тип int.

Пример 1. По запуску макрокоманды №1 последовательно воспроизводить с помощью аудиопроигрывателя № 1 звуковые файлы cam_alarm_1.wav, cam_alarm_2.wav, cam_alarm_3.wav из папки …\Intellect\Wav\. Задержка между началом воспроизведения каждого последующего звукового файла должна составлять 5 секунд (5000 миллисекунд).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")

{
 var i;
 for(i=1; i<=3; i=i+1)
 {
  DoReactStr("PLAYER", "1", "PLAY_WAV", "file<\cam_alarm_" + i + ".wav>");
  Sleep(5000);
 }
}
```

Пример 2. По запуску макрокоманды №2 запускается таймер №1, срабатывающий через каждые 10 секунд в течение 1 минуты с момента запуска макрокоманды №2.

Примечание.
Для запуска данного скрипта необходимо предварительно создать объект **Таймер** с идентификационным номером 1. Параметры объекта следует оставить установленными по умолчанию (**Любой(-ая)**). Объект **Таймер 1** может быть отключен.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "2" && Event.Action == "RUN")

{
 for(i=0; i<=5; i=i+1)

 {
  DoReactStr("TIMER","1", "DISABLE", "");
  Sleep(10000);
  DoReactStr("TIMER","1", "ENABLE", "");

  NotifyEventStr("TIMER","1", "TRIGGER", "");
 }
 DoReactStr("TIMER","1", "DISABLE", "");
}
```

Метод Itv_var

Метод Itv_var используется для задания и возвращения значений глобальных переменных.

Синтаксис обращения к методу:

```
function  Itv_var (globalvar : String) : String
```

Аргументы метода:

1. **globalvar** – обязательный аргумент. Задает название глобальной переменной. Допустимые значения: тип String, удовлетворяющие требованиям к допустимым названиям строковых (String) параметров системного реестра ОС Windows.

Примечание.
Глобальные переменные хранятся в системном реестре, что обеспечивает сохранность их значений после перезапуска ОС Windows. Все глобальные переменные хранятся в ветвях реестра HKEY_USERS\S-1-5-21-…\Software\ITVScript\ITVSCRIPT и HKEY_CURRENT_USER\Software\ITVScript\ ITVSCRIPT. Для доступа к глобальной переменной непосредственно из реестра требуется осуществить поиск по ее названию.

Пример. По запуску макрокоманды №1 сохранять значение параметра **Яркость** (“bright”) для камеры №10 в глобальную переменную cam10bright. По запуску макрокоманды №2 устанавливать камерам 1-4 значение параметра **Яркость** равным значению глобальной переменной cam10bright.

```
if (Event.SourceType == "MACRO" && Event.Action == "RUN")

{
 if(Event.SourceId == "1")
 {
  Itv_var("cam10bright") = GetObjectParam("CAM", "10", "bright");
 }
 if (Event.SourceId == "2")
 {
  var cam10bright = Itv_var("cam10bright");
  for(i=1; i<=4; i=i+1)
  {
   SetObjectParam("CAM", i, "bright", cam10bright);
  }
 }
}
```

Метод Int_var

Метод Int_var используется для задания и возвращения значений глобальных переменных целочисленного типа.

Внимание!
Метод Int_var использует то же хранилище, что и Метод Itv_var, но приводит тип переменной к целочисленному.

Синтаксис обращения к методу:

```
function  Int_var (globalvar : String) : int
```

Аргументы метода:

1. **globalvar** – обязательный аргумент. Задает название глобальной переменной. Допустимые значения: тип String, удовлетворяющие требованиям к допустимым названиям строковых (String) параметров системного реестра ОС Windows.

Примечание.
Глобальные переменные хранятся в системном реестре, что обеспечивает сохранность их значений после перезапуска ОС Windows. Все глобальные переменные хранятся в ветвях реестра HKEY_USERS\S-1-5-21-…\Software\ITVScript\ITVSCRIPT и HKEY_CURRENT_USER\Software\ITVScript\ ITVSCRIPT. Для доступа к глобальной переменной непосредственно из реестра требуется осуществить поиск по ее названию.

Пример. В приведенном ниже тестовом примере для проверки работы метода глобальной переменной с названием "2" присваивается значение 1, которое затем увеличивается на единицу и выводится в отладочное окно скрипта.

```
if(Event.Action =="RUN")
{
   Int_var(2) = 1;
   Int_var("2")++;
   DebugLogString(Int_var("2").toString());
}
```

Метод GetObjectParentType

Метод GetObjectParentType возвращает тип родительского объекта для заданного объекта в соответствии предусмотренной в системе иерархией системных объектов.

Синтаксис обращения к методу:

```
function  GetObjectParentType (objtype : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, для которого требуется возвратить тип родительского объекта. Допустимые значения: значения типа String, диапазон ограничен допустимыми в системе типами объектов.

Примечание.
В иерархии системных объектов самым старшим является объект Main. Данный объект является родительским для всех объектов типа Computer («Компьютер»), Screen («Экран») и проч.

Пример. По запуску макрокоманды № 1 отобразить в отладочном окне названия четырех объектов, начиная с зоны детектора, в порядке предусмотренной ПК *Интеллект* иерархии.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
 var objtype = "CAM_ZONE";
 DebugLogString(objtype);
 for(var i = 1; i<=4; i=i+1)
 {
  objtype = GetObjectParentType(objtype);
  DebugLogString(objtype);
 }
}
```

Метод GetIPAddress

Метод GetIPAddress возвращает IP-адрес соединения ядер программного комплекса *Интеллект* в соответствии с существующей архитектурой распределенной системы видеонаблюдения.

Синтаксис обращения к методу:

```
function  GetIPAddress (dst : String, src : String) : String
```

Аргументы метода:

1. **dst** – обязательный аргумент. Задает наименование удаленного компьютера, на котором установлено ядро программного комплекса *Интеллект*. Значение аргумента **dst** должно совпадать с одним из наименований компьютеров, зарегистрированных при настройке архитектуры распределенной системы видеонаблюдения. Допустимые значения: значения типа String, удовлетворяющие требованиям к сетевым именам компьютеров; диапазон ограничен зарегистрированными в системе наименованиями компьютеров.
2. **src** – обязательный аргумент. Задает наименование локального компьютера (компьютера, с которого производится запуск скрипта). Значение аргумента **src** должно совпадать наименованием локального компьютера, под которым он зарегистрирован в программном комплексе *Интеллект*. Допустимые значения: значения типа String, удовлетворяющие требованиям к сетевым именам компьютеров.

Примечание.
Данные обо всех зарегистрированных при настройке распределенной архитектуры соединениях локального компьютера (ядра) с другими удаленными компьютерами (ядрами), отображаются во вкладке **Архитектура** диалогового окна **Настройка системы**.

Пример. По тревоге от камеры определить имя компьютера, к которому подключена данная камера, и вывести в отладочное окно IP-адрес соединения данного компьютера с локальным (на котором работает скрипт).

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")

{
 var camid = Event.SourceId;
 var compname = GetObjectParentId("CAM", camid, "COMPUTER");\\определение имени компьютера, к которому подключена тревожная камера
 var ip = GetIPAddress(compname,"WS1"); \\ определение IP-адреса соединения с компьютером, на котором установлена тревожная камера
 DebugLogString("IP-адрес соединения с компьютером, на котором установлена тревожная камера  " + ip);
}
```

Примечание.
Вместо “WS1” необходимо вписать имя компьютера, на котором запускается скрипт, и установлено ядро ПК *Интеллект*.

Метод GetObjectName

Метод GetObjectName возвращает название объекта, заданное ему при регистрации в программном комплексе.

Синтаксис обращения к методу:

```
function  GetObjectName(objtype : String, id : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Задает системный тип объекта, название которого требуется получить. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.

Пример. При тревоге по любому лучу вызывать диалоговое окно с информационным сообщением: «Тревога по Лучу (название луча, по которому зарегистрирована тревога). Луч подключен к Серверу (название раздела, к которому относится тревожный луч)».

Примечание.
Предварительно необходимо с помощью утилиты *Arpedit.exe* создать диалоговое окно и сохранить его в файле test.dlg в папке <Директория установки ПК *Интеллект*> \Program.

```
if (Event.SourceType == "GRAY" && Event.Action == "ALARM")
{
 var grayid = Event.SourceId;
 var grayname = GetObjectName("GRAY", grayid);
 var compname = GetObjectParentId("GRAY", grayid, "COMPUTER");
 DoReactStr("DIALOG", "test", "CLOSE_ALL","");
 DoReactStr("DIALOG", "test", "RUN","Тревога по Лучу `" + grayname + "'. Луч подключен к Серверу `" + compname + "'.");
}
```

Метод GetObjectState

Метод GetObjectState возвращает состояние системного объекта на момент обращения.

Синтаксис обращения к методу:

```
function  GetObjectState(objtype : String, id : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Задает тип системного объекта, состояние которого требуется получить. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.

Пример. При включении реле №1 (например, нажатии кнопки, подключенной к реле №1) поставить на охрану луч №1. При повторном включении реле №1, снять с охраны луч № 1.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1" && Event.Action == "ON")
{
 if(GetObjectState("GRAY", "1")=="DISARM")
 {
  SetObjectState("GRAY", "1", "ARM");
 }
 else
 {
  SetObjectState("GRAY", "1", "DISARM");
 }
}
```

Метод GetObjectParam

Метод GetObjectParam возвращает значение заданного параметра системного объекта на момент обращения.

Синтаксис обращения к методу:

```
function  GetObjectParam(objtype : String, id : String, param : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Задает тип системного объекта, для которого требуется возвратить значение заданного параметра. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **param** – обязательный аргумент. Соответствует названию параметра, значение которого требуется возвратить. Допустимые значения: тип String, диапазон ограничен допустимыми для заданного объекта параметрами.

Пример. См. пример в разделе Метод SetObjectParam.

Метод GetObjectParentId

Метод GetObjectParentId возвращает идентификационный (регистрационный) номер родительского объекта для заданного объекта.

Синтаксис обращения к методу:

```
function  GetObjectParentId(objtype : String, id : String, parent : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Задает тип системного объекта, для которого требуется возвратить тип родительского объекта. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **parent** – обязательный аргумент. Задает тип системного объекта, родительского (т.е. старшего в соответствии с иерархией системных объектов) по отношению к объекту заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.

Пример. При отключении любой из камер системы или прекращении поступления видеосигнала с камеры отправлять почтовые сообщение, зарегистрированное в ПК *Интеллект* под номером 1. Сообщение должно включать тему «Внимание! Отключение камеры» и, в теле сообщения, информацию о номере отключенной камеры и номере сервера, на котором она установлена.

Примечание.
Предполагается, что *Сервис почтовых сообщений* настроен и корректно функционирует.

```
if ((Event.SourceType == "CAM" && Event.Action == "DETACH") || (Event.Action == "REC_STOP"))

{

 var cam_id = Event.SourceId;

 var parent_comp_id = GetObjectParentId("CAM", cam_id, "COMPUTER");

 DoReactStr("MAIL_MESSAGE","1","SETUP","from<***@mail.ru>,to<***@mail.ru>,body<Отключение камеры "+cam_id+" на сервере
"+parent_comp_id+">,parent_id<1>,subject<Внимание! Отключение камеры>,name<Почтовое сообщение 1>,objname<Почтовое сообщение 1>");

 DoReactStr("MAIL_MESSAGE", "1", "SEND", "");

}
```

Метод DoReactStr

Метод DoReactStr используется для генерации реакций системных объектов. Метод DoReactStr отправляет реакцию заданному объекту. При этом реакция передается непосредственно тому ядру, на котором зарегистрирован объект, а не всей системе. В методе DoReactStr реакция задается группой аргументов типа String.

Синтаксис обращения к методу:

```
function  DoReactStr(objtype : String,  id : String,  action : String,  param<value>  [, param<value>] : String)
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, для которого требуется генерировать реакцию. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **action** – обязательный аргумент. Задает реакцию, которую требуется генерировать. Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа реакциями.
4. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) реакции системного объекта.

Синтаксис задания значения одному параметру соответствует строке:

"param<value>", где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

"param1<value1>,param2<value2>…".

Список оформляется через запятую без пробелов. В том случае, если ни один параметр задавать не требуется, при обращении необходимо указать пустую строку, например:

```
DoReactStr("CAM","1","MD_START","");
```

Допустимые значения аргумента **param**: тип String, диапазон ограничен допустимыми для заданной реакции параметрами. Допустимые значения аргумента **value**: тип String, диапазон зависит от устанавливаемого параметра.

Для всех реакций имеется возможность указать задержку выполнения реакции при помощи параметра delay<>. Задержка указывается в секундах.

Примечание
В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса *Интеллект*, соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример. При регистрации тревоги по любой из видеокамер системы, переводить Монитор № 1 в режим отображения одного окна видеонаблюдения (однократор) и выводить в данном окне видеосигнал с камеры, по которой зарегистрирована тревога.

```
if (Event.SourceType == "CAM" &&  Event.Action == "MD_START")
{
 var camid = Event.SourceId;
 DoReactStr("MONITOR","1","ACTIVATE_CAM","cam<"+ camid +">");
 DoReactStr("MONITOR","1","KEY_PRESSED","key<SCREEN.1>");
}
```

Пример. При окончании тревоги по любой из видеокамер продолжать запись по ней в течении еще 5 секунд, после чего прекращать запись (аналог режима Дозапись).

```
 if (Event.SourceType == "CAM" &&  Event.Action == "MD_STOP")
{
 var camid = Event.SourceId;
 DoReactStr("CAM",camid,"REC_STOP","delay<5>");
}
```

Пример. По макрокоманде 1 включать управление телеметрией при помощи мыши на камере 4, выведенной на монитор 10, по макрокоманде 2 отключать.

```
if (Event.SourceType == "MACRO" &&  Event.Action == "RUN" &&  EventSourceId == "1")
{
 DoReactStr("MONITOR","10","CONTROL_TELEMETRY","cam<4>,on<1>");
}
if (Event.SourceType == "MACRO" &&  Event.Action == "RUN" &&  EventSourceId == "2")
{
 DoReactStr("MONITOR","10","CONTROL_TELEMETRY","cam<4>,on<0>");
}
```

Метод DoReact

Метод DoReact используется для генерации реакций системных объектов. Метод DoReact отправляет реакцию заданному объекту. При этом реакция передается непосредственно тому ядру, на котором зарегистрирован объект, а не всей системе. В методе DoReact реакция задается объектом **MsgObject**.

Синтаксис обращения к методу:

```
function  DoReact(msgevent : MsgObject)
```

Аргументы метода:

1. **msgevent** – обязательный аргумент. Задает реакцию, отправляемую заданному объекту. Допустимые значения: объекты **MsgObject**, ранее созданные в скрипте.

Примечание.
В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса *Интеллект*, соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример. По замыканию реле №1 замыкать также реле №2 и 3. По размыканию реле №1 размыкать также реле №2.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1")
{
 var msgevent = Event.Clone();
 if(Event.Action == "ON")
 {
  msgevent.SourceId = "2";
  DoReact(msgevent);
  msgevent.SourceId = "3";
  DoReact(msgevent);
 }
 if(Event.Action == "OFF")
 {
  msgevent.SourceId = "2";
  DoReact(msgevent);
 }
}
```

Метод DoReactSetupCore

Метод DoReactSetupCore предназначен для изменения параметров системного объекта. Данный метод изменяет только заданные параметры объекта, остальные оставляя без изменения.

Синтаксис обращения к методу:

```
function  DoReactSetupCore(objtype : String,  id : String, param<value>  [, param<value>] : String )
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, параметрам которого требуется задать значения. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) реакции системного объекта.

Синтаксис задания значения одному параметру соответствует строке:

“param<value>”, где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

“param1<value1>,param2<value2>…”.

Список оформляется через запятую без пробелов.

Допустимые значения аргумента **param**: значения типа String, диапазон ограничен допустимыми для заданной реакции параметрами. Допустимые значения аргумента **value**: значения типа String, диапазон зависит от устанавливаемого параметра.

Пример. По макрокоманде 1 установить камерам №1 – 4 новые значения параметров номер поворотного устройства (telemetry_id), номер микрофона для синхронной записи (audio_id<>). Значения должны быть на единицу больше, чем номера соответствующих камер.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
 var i;
 for(i=1; i<=4; i=i+1)
 {
  DoReactSetupCore("CAM", i, "telemetry_id<" + (i+1) +">,audio_id<" + (i+1) + ">");
 }
}
```

Метод DoReactSetup

Метод DoReactSetup предназначен для временного изменения параметров системного объекта. Данный метод изменяет только заданные параметры объекта, остальные оставляя без изменения.

Синтаксис обращения к методу:

```
function  DoReactSetup (objtype : String,  id : String, param<value>  [, param<value>] : String )
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, параметрам которого требуется задать значения. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) реакции системного объекта.

Синтаксис задания значения одному параметру соответствует строке:

“param<value>”, где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

“param1<value1>,param2<value2>…”.

Список оформляется через запятую без пробелов.

Допустимые значения аргумента **param**: значения типа String, диапазон ограничен допустимыми для заданной реакции параметрами. Допустимые значения аргумента **value**: значения типа String, диапазон зависит от устанавливаемого параметра.

Пример. По макрокоманде 1 временно удалить все камеры с первого монитора.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
 DoReactSetup ("MONITOR","1","REMOVE_ALL","");
}
```

Метод DoReactGlobal

Метод DoReactGlobal используется для генерации реакций системных объектов. Метод DoReactGlobal отправляет реакцию заданному объекту. При этом реакция передается не только тому ядру, на котором зарегистрирован объект, но и всей системе. В методе DoReactGlobal реакция задается объектом **MsgObject**.

Синтаксис обращения к методу:

```
function  DoReactGlobal(msgevent : MsgObject)
```

Аргументы метода:

1. **msgevent** – обязательный аргумент. Задает реакцию, отправляемую заданному объекту. Допустимые значения: объекты **MsgObject**, ранее созданные в скрипте.

Пример. При выполнении макрокоманды №2, ставить луч №2 на охрану. Команду отправлять по всем ядрам системы в виде реакции для регистрации в Протоколе событий.

```
if (Event.SourceType == "MACRO"&& Event.SourceId == "2" && Event.Action == "RUN")
{
 var msgevent = CreateMsg();
 msgevent.SourceType = "GRAY";
 msgevent.SourceId = "2";
 msgevent.Action = "ARM";
 DoReactGlobal(msgevent);
}
```

Метод NotifyEventStr

Метод NotifyEventStr используется для генерации системных событий. При этом генерируемое событие рассылается по всем ядрам системы, соединенным с локальным ядром. В методе NotifyEventStr событие задается группой аргументов типа String.

Синтаксис обращения к методу:

```
function  NotifyEventStr(objtype : String, id  : String, action : String, param<value>  [, param<value>] : String )
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, для которого требуется генерировать событие. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **action** – обязательный аргумент. Задает событие, которое требуется генерировать. Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа событиями.
4. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) системного события.

Синтаксис задания значения одному параметру соответствует строке:

"param<value>", где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

"param1<value1>,param2<value2>…".

Список оформляется через запятую без пробелов. В том случае, если ни один параметр задавать не требуется, при обращении необходимо указать пустую строку, например:

```
DoReactStr("CAM","1","MD_START","");
```

Допустимые значения аргумента **param**: тип String, диапазон ограничен допустимыми для заданного события параметрами. Допустимые значения аргумента **value**: тип String, диапазон зависит от устанавливаемого параметра.

Примечание
В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса Интеллект, соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример. При регистрации тревоги по камере отправлять в систему событие: «тревожное блокирование» для соответствующего камере раздела. Если идентификационный номер тревожной камеры лежит в диапазоне от 1 до 4 – для раздела № 1, если от 5 до 10 – для раздела № 2.

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{
 var regionid;
 if (Event.SourceId <=4)
 {
  regionid = "1";
 }
 if ((Event.SourceId > 4) &&  (Event.SourceId <= 10))
 {
  regionid = "2";
 }
 NotifyEventStr("REGION", regionid, "PANIC_LOCK", "");
}
```

Метод NotifyEvent

Метод NotifyEvent используется для генерации системных событий. При этом генерируемое событие рассылается по всем ядрам системы, соединенным с локальным ядром. В методе NotifyEvent событие задается объектом **MsgObject**.

Синтаксис обращения к методу:

```
function  NotifyEvent(msgevent : MsgObject)
```

Аргументы метода:

1. **msgevent** – обязательный аргумент. Задает событие, отправляемое в систему. Допустимые значения: объекты **MsgObject**, ранее созданные в скрипте.

Примечание.
Примечание
В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса *Интеллект*, соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример. По началу архивации видеозаписей модулем **Долговременный архив** № 1 отключается аналоговый выход № 1 платы видеоввода № 2. Необходимо отправлять в систему команду в виде события для регистрации в **Протоколе событий**.

Примечание.
При выполнении данного скрипта отключение аналогового выхода № 1 платы видеоввода № 2 не произойдет.

```
if (Event.SourceType == "ARCH" && Event.SourceId == "1" && Event.Action == "ACTIVE")
{
 var msgevent = CreateMsg();
 msgevent.SourceType = "GRABBER";
 msgevent.SourceId = "2";
 msgevent.Action = "MUX1_OFF";
 NotifyEvent(msgevent);
}
```

Метод NotifyEventGlobal

Метод NotifyEventGlobal используется для генерации системных событий. При этом генерируемое событие рассылается по всем ядрам системы, соединенным по сети. В методе NotifyEventGlobal событие задается объектом **MsgObject**.

Синтаксис обращения к методу:

```
function  NotifyEventGlobal (msgevent : MsgObject)
```

Аргументы метода:

1. **msgevent** – обязательный аргумент. Задает событие, отправляемое в систему. Допустимые значения: объекты **MsgObject**, ранее созданные в скрипте.

Пример. При выполнении макрокоманды №1 первая камера ставится на запись. Необходимо отправлять команду по всем ядрам системы в виде события для регистрации в Протоколе событий.

Примечание.
При выполнении данного скрипта не произойдет постановки камеры №1 на запись.

```
if (Event.SourceType == "MACRO"&& Event.SourceId == "1" && Event.Action == "RUN")
{
 var msgevent = CreateMsg();
 msgevent.SourceType = "CAM";
 msgevent.SourceId = "1";
 msgevent.Action = "REC";
 NotifyEventGlobal(msgevent);
}
```

Метод CreateMsg

Метод CreateMsg предназначен для создания объектов на основании прототипа **MsgObject**.

Синтаксис обращения к методу:

```
function CreateMsg() : MsgObject
```

Аргументы метода отсутствуют.

Пример 1. При регистрации тревоги по камере отправлять в систему событие: «тревожное блокирование» для соответствующего раздела. Если идентификационный номер тревожной камеры лежит в диапазоне от 1 до 4 – для раздела № 1, если от 5 до 10 – для раздела № 2.

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{

var msgevent = CreateMsg();
 msgevent.SourceType = "REGION";
 msgevent.Action = "PANIC_LOCK";
 if (Event.SourceId <=4)
 {
  msgevent.SourceId = "1";
 }
 if ((Event.SourceId > 4) &&  (Event.SourceId < 10))
 {
  msgevent.SourceId = "2";
 }
 NotifyEvent(msgevent);
}
```

Пример 2. При запуске таймера №1, через каждые 30 секунд, запускать макрокоманду №1.

Примечание.
Для запуска данного скрипта необходимо предварительно создать объект **Таймер** с идентификационным номером 1. Установить параметру **Секунда** объекта **Таймер** значение 1, остальные параметры оставить без изменений (по умолчанию **Любой(ая)**).

```
if (Event.SourceType == "TIMER" && Event.SourceId == "1" && Event.Action == "TRIGGER")
{
 var msg = CreateMsg();
 msg.StringToMsg(GetObjectParams("TIMER", "1"));
 if(msg.GetParam("s") == "1")
 {
  DoReactStr("MACRO", "1", "RUN", "");
  SetObjectParam("TIMER","1","s","30");
  DoReactStr("TIMER","1", "DISABLE", "");
  DoReactStr("TIMER","1", "ENABLE", "");
 }
 if(msg.GetParam("s") == "30")
 {
  DoReactStr("MACRO", "1", "RUN", "");
  SetObjectParam("TIMER","1","s","1");
  DoReactStr("TIMER","1", "DISABLE", "");
  DoReactStr("TIMER","1", "ENABLE", "");
 }
}
```

Методы Lock и Unlock

Методы Lock и Unlock используются для создания глобальной критической секции при необходимости обеспечения синхронизации скриптов, запускаемых в отдельных потоках. Метод Lock открывает критическую секцию, метод Unlock закрывает.

> ⚠ **Внимание!**
> Необходимо обязательно вызывать метод Unlock, если был вызван метод Lock. В противном случае система может зависнуть.

Рекомендуется по возможности избегать использования методов Lock и Unlock.

Синтаксис обращения к методам:

```
function Lock()
```

```
function Unlock()
```

Пример. По макрокоманде 1 посчитать общее количество лучей и реле, находящихся в тревоге. Подсчет объектов каждого типа производить одновременно (в отдельном скрипте). Результат записать в глобальную переменную counter.

Скрипт 1:

```
//Считается количество реле в тревоге
var i = Number(0);
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
 var msg = CreateMsg();
 msg.StringToMsg(GetObjectIds("GRELE"));
 var objCount = msg.GetParam("id.count");
 var k;
 for(k= 0; k < objCount; k++)
  if(GetObjectState("GRELE", msg.GetParam("id." + k))== "ALARM"){
   Lock();
   i = Itv_var("counter");
   i++;
   Itv_var("counter")=i;
   Unlock();
  }
}
```

Скрипт 2:

```
//Считается количество лучей в тревоге
var i = Number(0);
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
 var msg = CreateMsg();
 msg.StringToMsg(GetObjectIds("GRAY"));
 var objCount = msg.GetParam("id.count");
 var k;
 for(k = 0; k < objCount; k++)
  if(GetObjectState("GRAY", msg.GetParam("id." + k))== "ALARMED"){
   Lock();
   i = Itv_var("counter");
   i++;
   Itv_var("counter")=i;
   Unlock();
 }
}
```

Примечание.
Если в данном примере не использовать методы Lock() и Unlock(), могут возникнуть коллизии и посчитанное значение окажется меньше, чем реальное.

Метод IsAvailableObject

Метод IsAvailableObject используется для определения текущих прав доступа к объекту.

Синтаксис обращения к методу:

```
function  IsAvailableObject(compname: String, objtype: String,  id: String,  param : String) : String
```

Метод возвращает 0, если текущему пользователю не назначены права типа **param** на доступ к объекту, и 1, если назначены.

Аргументы метода:

1. **compname** – обязательный аргумент. Соответствует имени объекта **Компьютер**, на базе которого создан объект в дереве оборудования.
2. **objtype** – обязательный аргумент. Соответствует типу системного объекта, права доступа к которому требуется выяснить. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
3. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
4. **param** – обязательный аргумент. Соответствует номеру типа прав, наличие которых требуется выяснить. Описание прав приведено в разделе Ограничение прав администрирования, управления и мониторинга документа Руководство Администратора. Допустимые значения:
    a. 0 – права доступа rightsNoView. Метод вернет 1, если нет прав на администрирование, управление и мониторинг объекта (красный крестик).

b. 1 – права доступа rightsNoControl. Метод вернет 1, если есть права только на мониторинг объекта (буква M).
c. 2 – права доступа rightsViewAndControl. Метод вернет 1, если есть права на управление и мониторинг объекта (флажок зеленого цвета).
d. 3 – права доступа rightsViewOrControl. Метод вернет 1, если есть права на мониторинг или управление объектом.
e. 4 – права доступа rightsNot.
f. 5 – права доступа rightsConfigure. Метод вернет 1, если есть права на администрирование, управление и мониторинг объекта (флажок серого цвета).

Пример. В дереве оборудования на базе объекта **Компьютер** с именем "S-UYUTOVA" создан объект **Камера** с идентификатором 1. Выяснить текущие права на доступ к объекту.

```
var i = 0;
for(i = 0; i <= 5; i++)
{
  var result =
  IsAvailableObject('S-UYUTOVA','CAM','1', i);
  DebugLogString("right "+i+" = "+result);
}
```

Метод GetUserId

Метод GetUserId возвращает идентификатор текущего пользователя ПК *Интеллект*.

Синтаксис обращения к методу:

```
function  GetUserId (cmp : String) : String
```

Аргументы метода:

1. **cmp** – обязательный аргумент. Задает имя компьютера, на котором установлен программный комплекс *Интеллект*. Допустимые значения: значения типа String, удовлетворяющие требованиям к сетевым именам компьютеров; диапазон ограничен зарегистрированными в системе наименованиями компьютеров.

Пример. Вывести в отладочном окне идентификатор текущего пользователя программного комплекса *Интеллект,* установленного на компьютере 'WS3'.

```
DebugLogString(GetUserId("WS3"));
```

Метод GetEventDescription

Метод GetEventDescription используется для получения описания события на естественном языке.

Синтаксис обращения к методу:

```
function  GetEventDescription (obj_type : String, event : String)
```

Аргументы метода:

1. **obj_type** – обязательный аргумент. Задает тип объекта системы, описание события которого требуется получить.
2. **event** – обязательный аргумент. Задает название события, описание которого требуется получить.

Пример. При получении событий по камере 1 выводить сообщения об этом на естественном языке в отладочное окно.

```
if (Event.SourceType == "CAM"&& Event.SourceId == "1")
{
var str = GetEventDescription("CAM", Event.Action);
DebugLogString(str);
}
```

Метод GetObjectIdByParam

Метод GetObjectIdByParam позволяет получить идентификатор объекта, у которого некоторый параметр равен заданному значению. В случае, если таких объектов несколько, возвращается идентификатор первого найденного объекта. В случае, если таких объектов не найдено, возвращается 0.

Синтаксис обращения к методу:

```
function GetObjectIdByParam (obj_type : String, obj_param : String, param_value : String)
```

Аргументы метода:

1. **obj_type** – обязательный аргумент. Задает тип объекта системы, идентификатор которого требуется получить.
2. **obj_param** – обязательный аргумент. Задает название параметра в базе данных, по значению которого требуется искать объект.
3. **param_value** – обязательный аргумент. Задает требуемое значение параметра объекта.

Пример. Найти камеры, с которых поступает черно-белое изображение, и установить для них параметр **Цвет** (color) равным единице.

```
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
  var id = GetObjectIdByParam("CAM","color","0"); //получение идентификатора первого объекта
  while (id) //пока существуют объекты Камера, с которых поступает черно-белое изображение
  {
     SetObjectParam ("CAM", id, "color", "1"); //изменение параметра Цвет для найденного объекта
     id = GetObjectIdByParam("CAM","color","0"); //получение идентификатора следующего объекта
  }
}
```

Метод SaveToFile

Метод SaveToFile используется для сохранения в файл кадра с камеры, который поступает в параметре data события FRAME_SENT.

Синтаксис обращения к методу:

```
function  SaveToFile (path: String,  data: String,  param : Boolean)
```

Сохранение кадра также может осуществляться при помощи реакции GET_FRAME объекта CAM. Для этого необходимо указать в параметре path данной реакции путь для сохранения файла с кадром. Событие FRAME_SENT создается в системе, если у реакции GET_FRAME отсутствует параметр path. При этом в параметре data события FRAME_SENT хранится кадр видеоизображения, который требуется сохранить при помощи метода SaveToFile.

Данная реакция позволяет экспортировать кадр видеоизображения, даже если камера не отображается в окне Монитора видеонаблюдения.

Аргументы метода:

1. **path** – обязательный аргумент. Задает полный путь для сохранения файла с кадром.
2. **data** – обязательный аргумент. Задает данные для сохранения в файл.
3. **param** – обязательный аргумент. Определяет необходимость перекодирования из формата base64 перед сохранением. Возможные значения параметра:
    a. true – перед сохранением декодировать из base64;
    b. false – сохранить строку без перекодировки.

Время выполнения сохранения кадра зависит от частоты опорных кадров. Чем больше частота опорных кадров, тем меньше время.

Пример. В случае поступления кадра с Камеры 1 сохранить его в файл test.jpg на диске D:

```
if (Event.SourceType == "CAM" && Event.SourceId == "1" && Event.Action == "FRAME_SENT")
  {
    SaveToFile("D:\\test.jpg",Event.GetParam("data"),true);
  }
```

Метод GetLinkedObjects

Метод GetLinkedObjects используется для получения списка объектов, привязанных к заданной камере при помощи объекта **Связь объектов** (см. Руководство Администратора, Связь объектов с камерами).

Синтаксис обращения к методу:

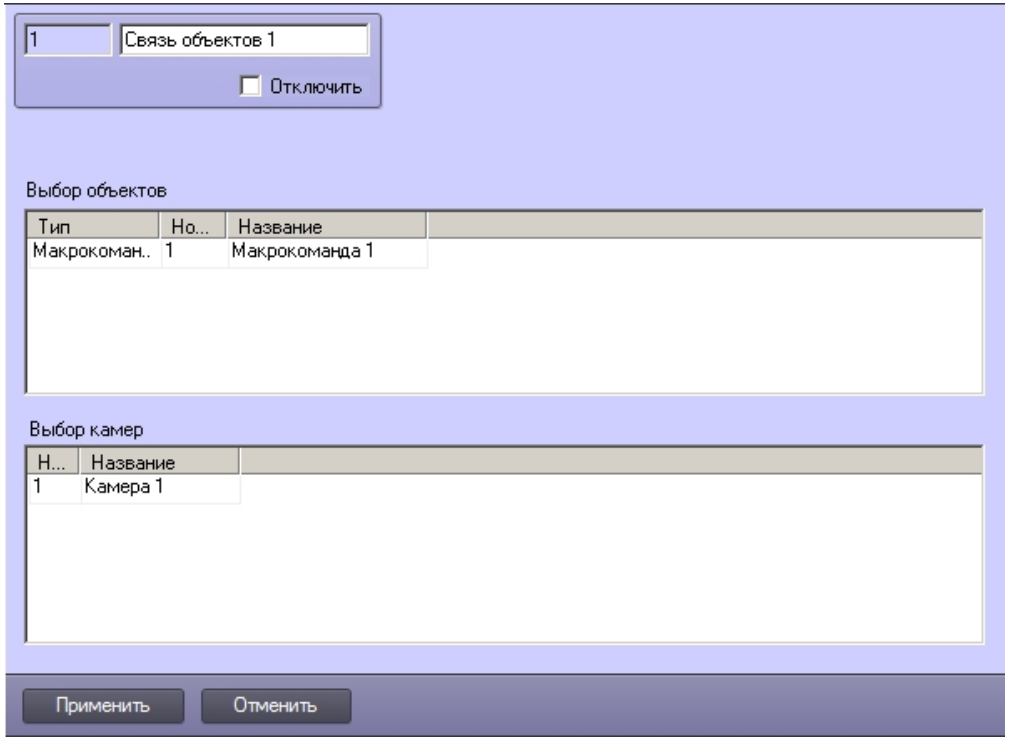
```
function  GetLinkedObjects (type1 : string, id : string, type2 : string)
```

Аргументы метода:

1. **type1** – тип объекта, для которого следует вернуть связанные.
2. **id** – идентификатор объекта, для которого следует вернуть связанные.
3. **type2** – тип связанных объектов, которые следует вернуть. Если передана пустая строка, будут возвращены связанные объекты всех типов.

Пример.

Объект **Связь объектов** настроен следующим образом:



Вывести в отладочное окно скрипта список объектов, связанных с камерой 1.

```
if (Event.SourceType == "MACRO")
{
   var msgstr = GetLinkedObjects("CAM","1","MACRO")
   DebugLogString("Связанные объекты  " + msgstr);
}
```

В результате в отладочное окно скрипта будет выведена строка «Связанные объекты MACRO:1».

Метод WriteIni

Метод WriteIni используется для записи строковой переменной в ini-файл.

Синтаксис обращения к методу:

```
function WriteIni(varName: String, varValue: String, path: String)
```

Аргументы метода:

1. varName – обязательный аргумент. Задает имя переменной для хранения в файле.
2. varValue – обязательный аргумент. Задает значение переменной.
3. path – обязательный аргумент. Задает полный путь к файлу ini, в котором должна храниться переменная. Хранилище переменных можно поместить на сетевом ресурсе, для этого в данном аргументе следует задать сетевой путь.

Пример. Записать переменную MyVar в файл \\fileserver\temp\test.ini, задав ей значение "Hello world!". Затем считать записанное значение и вывести его в отладочное окно скрипта.

```
WriteIni("MyVar", "Hello world", "\\\\fileserver\\temp\\test.ini");
var result = ReadIni("MyVar","\\\\fileserver\\temp\\test.ini");
DebugLogString(result);
```

Метод ReadIni

Метод ReadIni используется для чтения значения строковой переменной из ini-файла.

Синтаксис обращения к методу:

```
function ReadIni (varName: String, path: String): String
```

Аргументы метода:

1. varName – обязательный аргумент. Задает имя переменной, хранящейся в файле.
2. path – обязательный аргумент. Задает полный путь к файлу ini, в котором хранится переменная.

Пример см. в разделе Метод WriteIni.

Метод AddIni

Метод AddIni используется для записи, изменения и чтения значения целочисленной переменной из ini-файла. Метод возвращает значение переменной, полученное после изменения.

Синтаксис обращения к методу:

```
function AddIni(varName: String, varValue: int, path: String): int
```

1. varName – обязательный аргумент. Задает имя переменной в файле.
2. varValue – обязательный аргумент. Задает значение переменной либо значение, которое следует добавить к существующему значению переменной:
    a. Если в файле хранится переменная с именем varName и строковым значением, переменной будет присвоено значение varValue.
    b. Если в файле нет переменной с именем varName, будет создана такая переменная, и ей будет присвоено значение varValue.
    c. Если в файле хранится переменная с именем varName, которая имеет целочисленное значение, или же значение ее приводится к целочисленному типу, то значение будет приведено, и к нему будет прибавлено varValue.
3. path – обязательный аргумент. Задает полный путь к файлу ini, в котором должна храниться переменная. Хранилище переменных можно поместить на сетевом ресурсе, для этого следует задать сетевой путь.

Пример. В файле "C:\test.ini" нет переменной "MyVar". Записать в данный файл такую переменную со значением -1, прибавить к ней 1 и вывести полученное значение в отладочное окно скрипта.

```
var result = AddIni("MyVar", -1, "C:\\test.ini");

result = AddIni("MyVar", 1, "C:\\test.ini");

DebugLogString(result);
```

Объекты MsgObject и Event и их встроенные методы и свойства

Объекты MsgObject и Event

Объект **MsgObject** – это прототип (шаблон), используемый для создания объектов и реализующий используемые для обработки системных событий программного комплекса *Интеллект* методы и свойства. Методы и свойства объекта **MsgObject** позволяют получать сведения о системных объектах, от которых поступают события (или для которых инициализируются события), генерировать реакций для системных объектов, изменять их состояния и проч.

Обращение к методам и свойствам объекта-прототипа **MsgObject** осуществляется через объявленные и инициализированные на его основе объекты или через статический объект **Event**.

Объект **Event** – это статический объект, реализующий интерфейс обращения к системным событиям программного комплекса *Интеллект. Объект* **Event** обеспечивает доступ к системному событию, по которому был осуществлен запуск скрипта. При работе с объектом **Event** доступны все методы и свойства прототипа **MsgObject**.

Объявление (создание) объектов на основе прототипа **MsgObject** выполняется с использованием метода CreateMsg базового объекта **Core**.

Метод GetSourceType

Метод GetSourceType возвращает системный тип объекта **MsgObject** или **Event**.

Синтаксис обращения к методу:

```
function  GetSourceType() : String
```

Аргументы метода отсутствуют.

Пример. По макрокоманде № 1 ставить на охрану для камер № 1 – 4 зоны детекторов № *.1, настроенные на работу в режиме **День**. По макрокоманде № 2 ставить на охрану для камер № 1 – 4 зоны детекторов № *.2, настроенные на работу в режиме **Ночь**. По макрокоманде № 3 ставить на охрану для камер № 1 – 4 зоны детекторов № *.3, настроенные на работу в режиме **Осадки**.

Примечание.
Значок "*" соответствует идентификационному номеру видеокамеры в системе (от 1 до 4).

```
if(Event.GetSourceType() == "MACRO" && Event.GetAction() == "RUN")
{
 var k;
 //Перевод камер в режим работы "День" путем постановки на охрану зон детекторов № *.1
 if(Event.GetSourceId() == "1")
 {
  for(k=1; k<=4; k=k+1)
  {
   DoReactStr("CAM_ZONE", k + ".1", "ARM", "");
   DoReactStr("CAM_ZONE", k + ".2", "DISARM", "");
   DoReactStr("CAM_ZONE", k + ".3", "DISARM", "");
  }
 }
 //Перевод камер в режим работы "Ночь" путем постановки на охрану зон детекторов № *.2
 if(Event.GetSourceId() == "2")
 {
  for(k = 1; k <= 4; k = k+1)
  {
   DoReactStr("CAM_ZONE", k + ".1", "DISARM", "");
   DoReactStr("CAM_ZONE", k + ".2", "ARM", "");
   DoReactStr("CAM_ZONE", k + ".3", "DISARM", "");
  }
 }
 //Перевод камер в режим работы "Осадки" путем постановки на охрану зон детекторов № *.3
 if(Event.GetSourceId() == "3")
 {
  for(k = 1; k <= 4; k = k+1)
  {
   DoReactStr("CAM_ZONE", k + ".1", "DISARM", "");
   DoReactStr("CAM_ZONE", k + ".2", "DISARM", "");
   DoReactStr("CAM_ZONE", k + ".3", "ARM", "");
  }
 }
}
```

Метод GetSourceId

Метод GetSourceId возвращает идентификационный (регистрационный) номер объекта **MsgObject** или **Event**.

Синтаксис обращения к методу:

```
function  GetSourceId() : String
```

Аргументы метода отсутствуют.

Пример. См. пример в разделе Метод GetSourceType.

Метод GetAction

Метод GetAction возвращает событие, поступившее в форме объекта **Event**, или заданное для объекта **MsgObject**.

Синтаксис обращения к методу:

```
function  GetAction() : String
```

Аргументы метода отсутствуют.

Пример. См. пример в разделе Метод GetSourceType.

Метод GetParam

Метод GetParam возвращает значение заданного параметра системного объекта для объекта **MsgObject** или **Event**.

Синтаксис обращения к методу:

```
function  GetParam(param: String) : String
```

Аргументы метода:

1. **param** – обязательный аргумент. Соответствует названию параметра системного объекта, для которого создан объект **MsgObject** (или статический объект **Event**). Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа параметрами.

Примечание.
В случае если у объекта отсутствует параметр с таким названием, метод возвращает пустую строку.

Пример. По регистрации любого события от любой камеры проверять, с какого компьютера пришло данное событие. В том случае, если имя компьютера равно «WS3», создать копию события, в которой установить имя компьютера равным «Computer».

```
if (Event.SourceType == "CAM")
{
 var msg = Event.Clone();
 if (msg.GetParam("slave_id") == "WS3")
 {
  msg.SetParam("slave_id", "Computer");
  NotifyEvent(msg);
 }
}
```

Метод SetParam

Метод SetParam устанавливает значение заданному параметру объекта **MsgObject** или **Event**. Данный метод изменяет только заданные параметры объекта, остальные оставляя без изменения.

Синтаксис обращения к методу:

```
function  SetParam(param : String,  value : String)
```

Аргументы метода:

1. **param** – обязательный аргумент. Соответствует названию параметра системного объекта, для которого создан объект **MsgObject** (или статический объект **Event**). Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа параметрами.
2. **value** – обязательный аргумент. Устанавливает значение параметру, заданному аргументом **param**. Допустимые значения: тип String, диапазон зависит от устанавливаемого параметра.

Пример. См. пример в разделе Метод GetParam.

Метод MsgToString

Метод MsgToString преобразует объекты **MsgObject** (в том числе статический объект **Event**) в переменную типа String.

Синтаксис обращения к методу:

```
function  MsgToString() : String
```

Аргументы метода отсутствуют.

Пример. Отправлять сообщения обо всех событиях, зарегистрированных для микрофона №1, на заданный электронный почтовый ящик.

Примечание.
Предполагается, что **Сервис почтовых сообщений** настроен и корректно функционирует.

```
if (Event.SourceType == "OLXA_LINE" && Event.SourceId == "1")
{
 var msgstr = Event.MsgToString();
 DoReactStr("MAIL_MESSAGE", "1", "SETUP", "subject<Микрофон 1>,body<" + msgstr + ">");
 DoReactStr("MAIL_MESSAGE", "1", "SEND", "");
}
```

Метод StringToMsg

Метод StringToMsg преобразует переменную типа String в объект **MsgObject**.

Синтаксис обращения к методу:

```
function StringToMsg(msg : String) : MsgObject
```

Аргументы метода:

1. **msg** – обязательный аргумент. Задает переменную типа String, которую требуется преобразовать в объект **MsgObject**. Допустимые значения: переменные типа String, удовлетворяющие синтаксису представления объектов **MsgObject**:

"objtype|id|action|param1<value1>,param2<value2>…", где

objtype – тип системного объекта;

id – идентификационный номер системного объекта;

action – событие или реакция для системного объекта;

param1<value1>,param2<value2> - список параметров со значениями. Список оформляется через запятую без пробелов. В том случае, если ни один параметр задавать не требуется, необходимо оставить пустой строку после вертикальной черты «|», например:

"CAM|1|MD_START|"

Пример. По тревоге от лучей № 1 и 3 начинать запись аудиосигнала с микрофона № 1. По тревоге от лучей № 2 и 4 начинать запись аудиосигнала с микрофона № 2.

```
if (Event.SourceType == "GRAY" && Event.Action == "ALARM")
{
 var audioid;
 if (Event.SourceId == "1" || Event.SourceId == "3")
 {
  audioid = "1";
 }
 if (Event.SourceId == "2" || Event.SourceId == "4")
 {
  audioid = "2";
 }
 var str = "OLXA_LINE|"+audioid+"|ARM|";
 var msg = CreateMsg();
 msg.StringToMsg(str);
 NotifyEvent(msg);
}
```

Метод StringToParams

Метод StringToParams преобразует переменную типа String в список параметров и перезаписывает существующий список параметров объекта **MsgObject**.

Синтаксис обращения к методу:

```
function StringToParams(params: String)
```

Аргументы метода:

1. **params** – Обязательный аргумент. Задает переменную типа String, которую требуется преобразовать в список параметров объекта **MsgObject**. Допустимые значения: переменные типа String, удовлетворяющие синтаксису представления списка параметров объектов **MsgObject**:

"param1<value1>,param2<value2>…", где

param1<value1>,param2<value2> - список параметров со значениями. Список оформляется через запятую без пробелов. В том случае, если ни один параметр задавать не требуется, необходимо оставить пустой строку после вертикальной черты «|», например:

"CAM|1|MD_START|"

Пример. При регистрации события **Подключение** («attach») для любой из камер, требуется повторно инициировать событие **Подключение** в системе с измененными значениями параметров **Номер поворотного устройства** (telemetry_id) и **Номер микрофона для синхронной записи** (audio_id). Значения должны быть на единицу больше, чем номера соответствующих камер.

```
if (Event.SourceType == "CAM" && Event.Action == "ATTACH")
{
 var i;
 for (i=1;i<=4;i=i+1)
 {
  var msg = Event.Clone();
  var str = "telemetry_id<" + (i+1) +">,audio_id<" + (i+1) + ">";
  msg.StringToParams(str);
  NotifyEvent(msg);
 }
}
```

Метод Clone

Метод Clone создает копию объектов **MsgObject** и **Event**.

Синтаксис обращения к методу:

```
function Clone() : MsgObject
```

Аргументы метода отсутствуют.

Пример. По замыканию реле №1 начинать видеозапись по камере №1 и замыкать реле №2. По размыканию реле №1 начинать видеозапись по камере №2 и размыкать реле №2.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1")
{
 var msgevent = Event.Clone();
 if(Event.Action == "ON")
 {
  msgevent.SourceId = "2";
  DoReactStr("CAM","1","REC","");
  DoReactStr("GRELE","2","ON","");
 }
 if(Event.Action == "OFF")
 {
  msgevent.SourceId = "2";
  DoReactStr("CAM","2","REC","");
  DoReactStr("GRELE","2","OFF","");
 }
}
```

Метод GetObjectIds

Метод GetObjectIds отвечает за получение идентификаторов всех объектов определённого типа.

Синтаксис обращения к методу:

```
function GetObjectIds(objectType : String)
```

В ответ возвращается строка:

CAM||COUNT|id.3<5>,id.count<4>,id.0<2>,id.1<3>,id.2<4>

где id.count<> – количество ID объектов,

id.[число]<> – ID объекта.

Аргументы метода:

1. **objectType** – обязательный аргумент. Задает тип системного объекта, для которого требуется возвратить значение заданного параметра (**CAM**, **GRAY**, **GRABBER** и т.п.). Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.

Пример. По запуску Макрокоманды №1 необходимо поставить все камеры на охрану.

```
if (Event.SourceType == "MACRO" && Event.SourceId && Event.Action == "RUN")
{
 var msg = CreateMsg();
 msg.StringToMsg(GetObjectIds("CAM"));
 var objCount = msg.GetParam("id.count");
 var i;
 for(i = 0; i < objCount; i++)
 {
  DoReactStr("CAM", msg.GetParam("id." + i), "ARM", "");
 }
}
```

Метод GetObjectParams

Метод GetObjectParams используется для получения параметров объекта.

Синтаксис обращения к методу:

```
function GetObjectParams(objectType : String, objectId : String)
```

Аргументы метода:

1. **objectType** – обязательный аргумент. Задает тип системного объекта(**CAM**, **GRAY**, **GRABBER** и т.п.), для которого требуется возвратить тип родительского объекта. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **objectId** – идентификатор объекта. Допустимые значения: тип String.

Пример. По запуску Макрокоманды № 1 необходимо проверить цветность камеры №2. В том случае, если камера №2 цветная, поставить ее в режим записи.

```
if (Event.SourceType == "MACRO" && Event.SourceId && Event.Action == "RUN")
{
 var msg = CreateMsg();
 msg.StringToMsg(GetObjectParams("CAM", "2"));
 if(msg.GetParam("color") == "1")
 {
  DoReactStr("CAM", "2", "REC", "");
 }
}
```

Свойство SourceType

Свойство SourceType позволяет возвращать и устанавливать системный тип для объектов **MsgObject** и **Event**.

Синтаксис обращения к свойству:

```
SourceType : String
```

Пример. По замыканию реле № 1 (например, по нажатию подключенной к реле кнопки) печатать кадры с камер №1 и 2.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1" && Event.Action == "ON")
{
 //активирование окна видеонаблюдения камеры №1
 DoReactStr("MONITOR","1","ACTIVATE_CAM", "cam<1>");
 //печать кадра, экспортированного с камеры №1
 DoReactStr("MONITOR","1","KEY_PRESSED","key<PRINT>");
 //активирование окна видеонаблюдения камеры №2
 DoReactStr("MONITOR","1","ACTIVATE_CAM", "cam<2>");
 //печать кадра, экспортированного с камеры №2
 DoReactStr("MONITOR","1","KEY_PRESSED","key<PRINT>");
}
```

Свойство SourceId

Свойство SourceId позволяет возвращать и устанавливать идентификационный номер для объектов **MsgObject** и **Event**.

Синтаксис обращения к свойству:

```
SourceId : String
```

Пример. См. пример в разделе Свойство SourceType.

Свойство Action

Свойство Action позволяет возвращать и устанавливать реакцию или событие для объектов **MsgObject** и **Event**.

Синтаксис обращения к свойству

```
SourceId : String
```

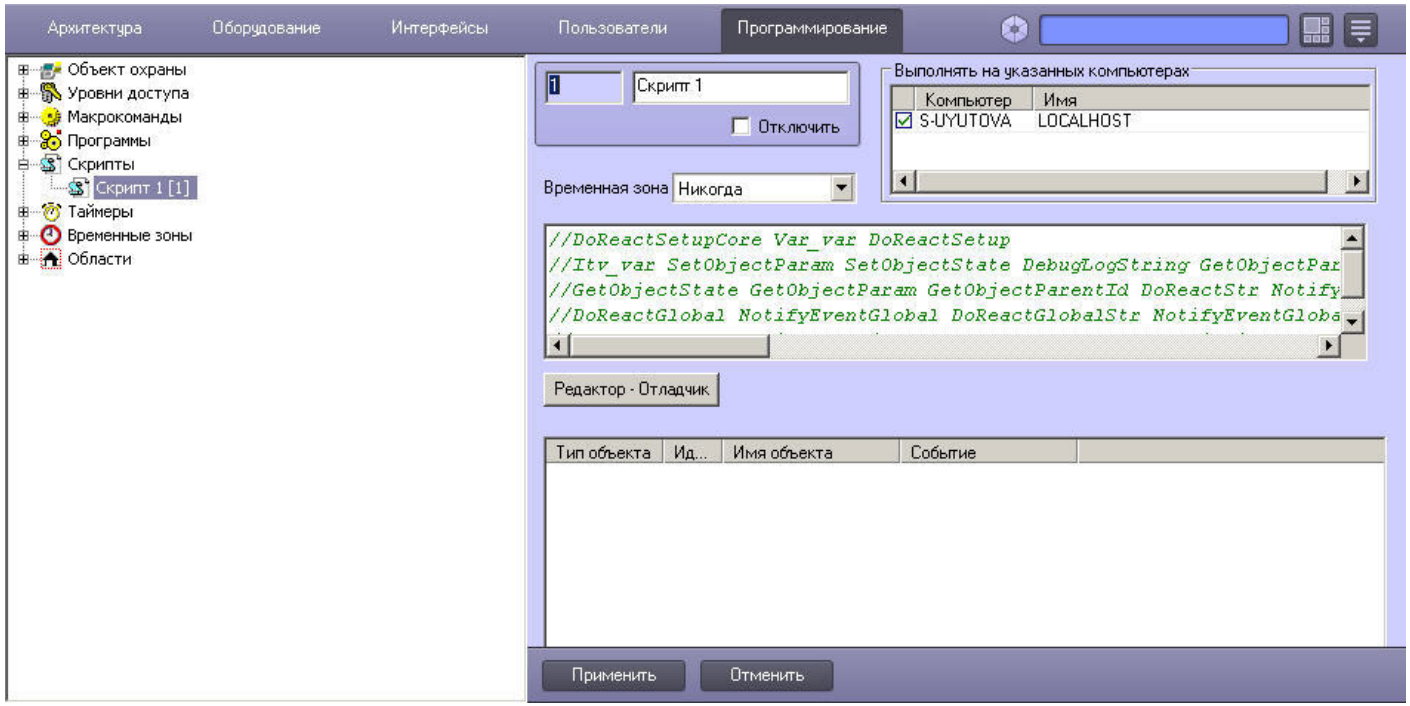
Пример. См. пример в разделе Свойство SourceType.

Инструментарий программирования на JScript

Системный объект Скрипт

Системный объект **Скрипт** предназначен для инициализации в ПК *Интеллект* скрипта, разработанного на языке JScript, и задания параметров его выполнения.

Панель настройки системного объекта **Скрипт** представлена на рисунке:



Внимание!
Создание большого количества (более 100) объектов **Скрипт** может привести к нестабильной работе системы.

В панели настройки системного объекта **Скрипт** указываются временная зона выполнения скрипта и компьютеры (ядра), на которых требуется выполнять скрипт.

На панели настройки системного объекта **Скрипт** размещены кнопка запуска утилиты *Редактор-Отладчик* и панель просмотра текста скрипта, созданного посредством данной утилиты. Редактирование скрипта может осуществляться с использованием утилиты *Редактор-Отладчик* или непосредственно из панели настройки объекта **Скрипт**.

Кроме того, имеется возможность настроить фильтр событий – список событий, которые будет обрабатывать системный объект **Скрипт**. Вообще говоря внесение события в фильтр равносильно оператору if в тексте скрипта, то есть при указании события в таблице данный оператор можно опустить.

Внимание!
Задание фильтра событий обязательно выполнять при создании скрипта в больших распределенных конфигурациях. В противном случае модуль будет обрабатывать все входящие события и может работать некорректно.

Пример.
Пусть в таблице в столбце **Тип объекта** указано значение **Макрокоманда**, в столбце **Идентификатор** выбрано значение **1**, в столбце **Событие** - **Выполнено действие**. Тогда вместо скрипта

```
if (Event.SourceType == "MACRO" && Event.SourceId==1 && Event.Action == "RUN")
{
 DoReactStr("CAM", "2", "REC", "");
}
```

можно использовать скрипт

```
DoReactStr("CAM", "2", "REC", "");
```

Подробно элементы панели настроек объекта **Скрипт** описаны в документе Руководство администратора.

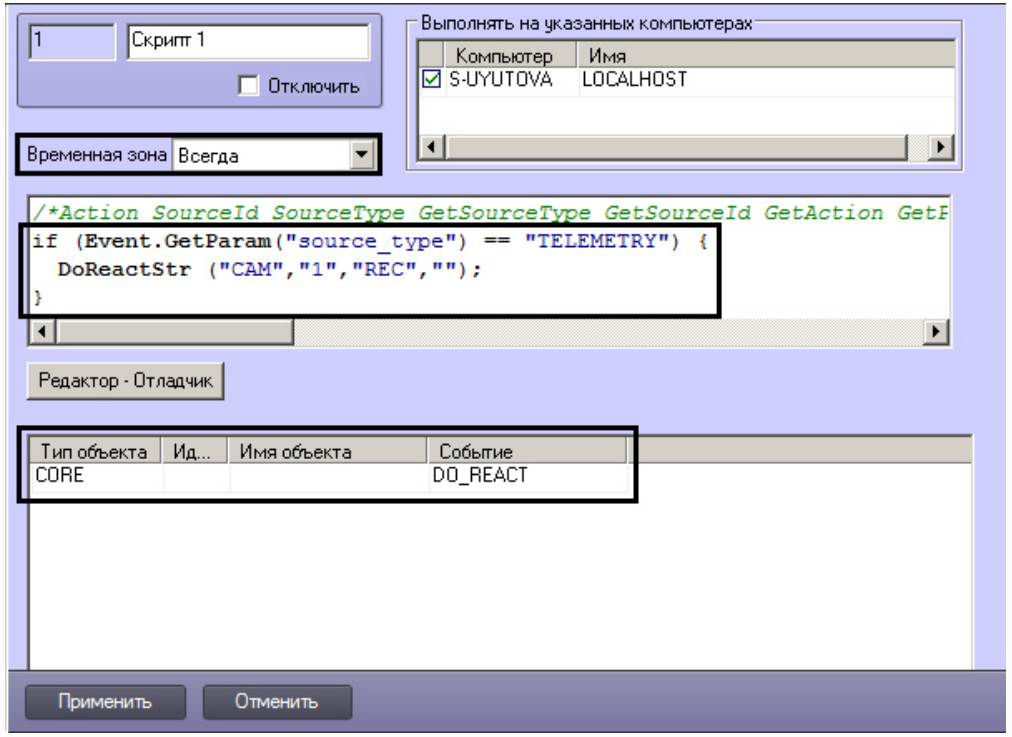
Пример.
При управлении поворотной камерой из Монитора видеонаблюдения начинать запись по Камере 1.

Для этого следует настроить объект **Скрипт** следующим образом:

1. Выбрать требуемую временную зону, когда должен выполняться скрипт.
2. Ввести текст скрипта:

```
if (Event.GetParam("source_type") == "TELEMETRY") {
  DoReactStr ("CAM","1","REC","");
}
```

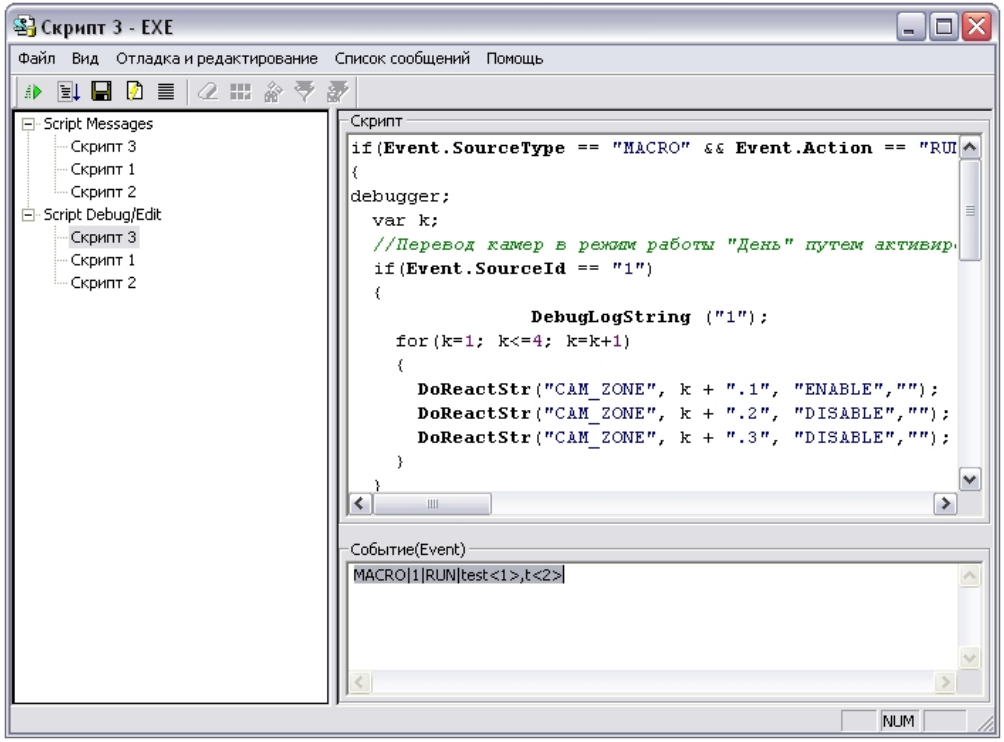
3. Настроить фильтр следующим образом:
   a. Из раскрывающегося списка **Тип объекта** выбрать CORE.
   b. В поле **Событие** ввести DO_REACT.



Утилита Редактор-Отладчик

Утилита *Редактор-Отладчик* предназначена для создания, отладки и редактирования скриптов в программном комплексе *Интеллект*.

Диалоговое окно утилиты *Редактор-Отладчик* представлено на рисунке.



Утилита *Редактор-Отладчик* содержит встроенные текстовый редактор и отладочное окно.

Для удобства контроля корректности написания скриптов в текстовом редакторе реализовано автоматическое выделение объектов, методов и свойств, входящих во встроенную в программный комплекс *Интеллект* объектную модель языка JScript.

Отладочное окно утилиты *Редактор-Отладчик* позволяет просматривать сведения обо всех событиях, регистрируемых в системе. Имеется возможность настроить фильтр событий, отображающихся в отладочном окне. Для каждого системного объекта **Скрипт** в утилите *Редактор-Отладчик* создается отдельное отладочное окно, что при использовании фильтров дает возможность отлаживать каждый скрипт независимо от других.

Для отладки скрипта предусмотрена возможность тестового запуска с использованием заданного пользователем тестового события, генерируемого утилитой и не регистрируемого в системе.

Созданный скрипт может быть сохранен в системном объекте **Скрипт** или в текстовом файле на жестком диске компьютера.

Отладочное окно

В программном комплексе *Интеллект* существует возможность в реальном времени просматривать все события и реакции, происходящие в системе. События и реакции со свойствами объектов отображаются в **Отладочном окне**, откуда их можно скопировать в буфер обмена Windows для последующего использования в программах.

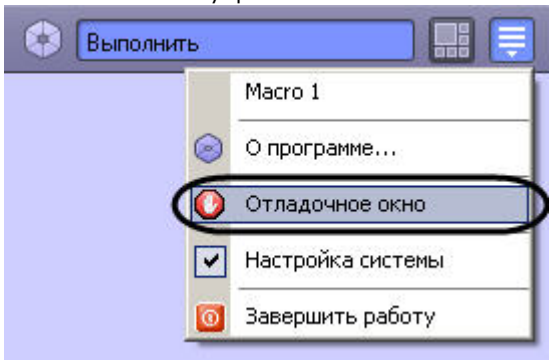
Включение Отладочного окна

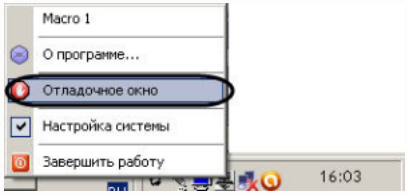
Отладочное окно по умолчанию выключено. Для включения Отладочного окна необходимо выполнить следующую последовательность действий:

1. Завершить работу программного комплекса *Интеллект.*
2. Запустить утилиту **Расширенная настройка***Tweaki.exe*.

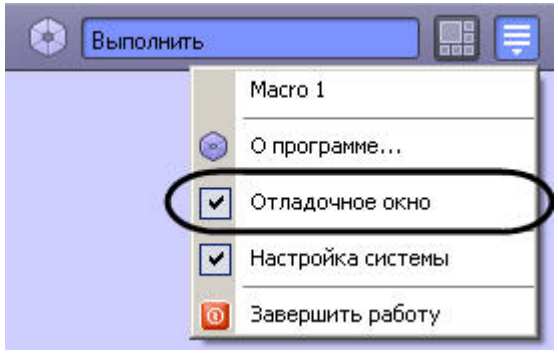
> **Примечание.**
> Отладочное окно можно включить и без использования утилиты tweaki.exe. Для этого следует установить строковый параметр «Debug» равным 1, 2, либо 3 в разделе «HKEY_LOCAL_MACHINE\SOFTWARE\ITV\Intellect» реестра ОС Windows (HKEY_LOCAL_MACHINE \Software\Wow6432Node\ITV\Intellect для 64-битной системы).

3. Выбрать раздел **Интеллект** в дереве, расположенном в левой части диалогового окна утилиты.
4. Изменить значение параметра **Режим отладки** с **Выключен** на **Debug 1**, **Debug 2** или **Debug 3**.
5. Нажать кнопку **ОК**.
6. Запустить программный комплекс *Интеллект*.
7. В Главной панели управления ПК *Интеллект* появится новый пункт **Отладочное окно**.



> **Примечание.**
>
> Также данное меню доступно из области уведомлений (системного трея) Windows при нажатии левой кнопкой мыши на значок  или по кратковременному удержанию горячей клавиши F8.
>
> 
>

8. Выбрать пункт **Отладочное окно** в Главной панели управления для отображения Отладочного окна на экране монитора. Выбранный пункт меню **Отладочное окно** бу дет отмечен флажком .

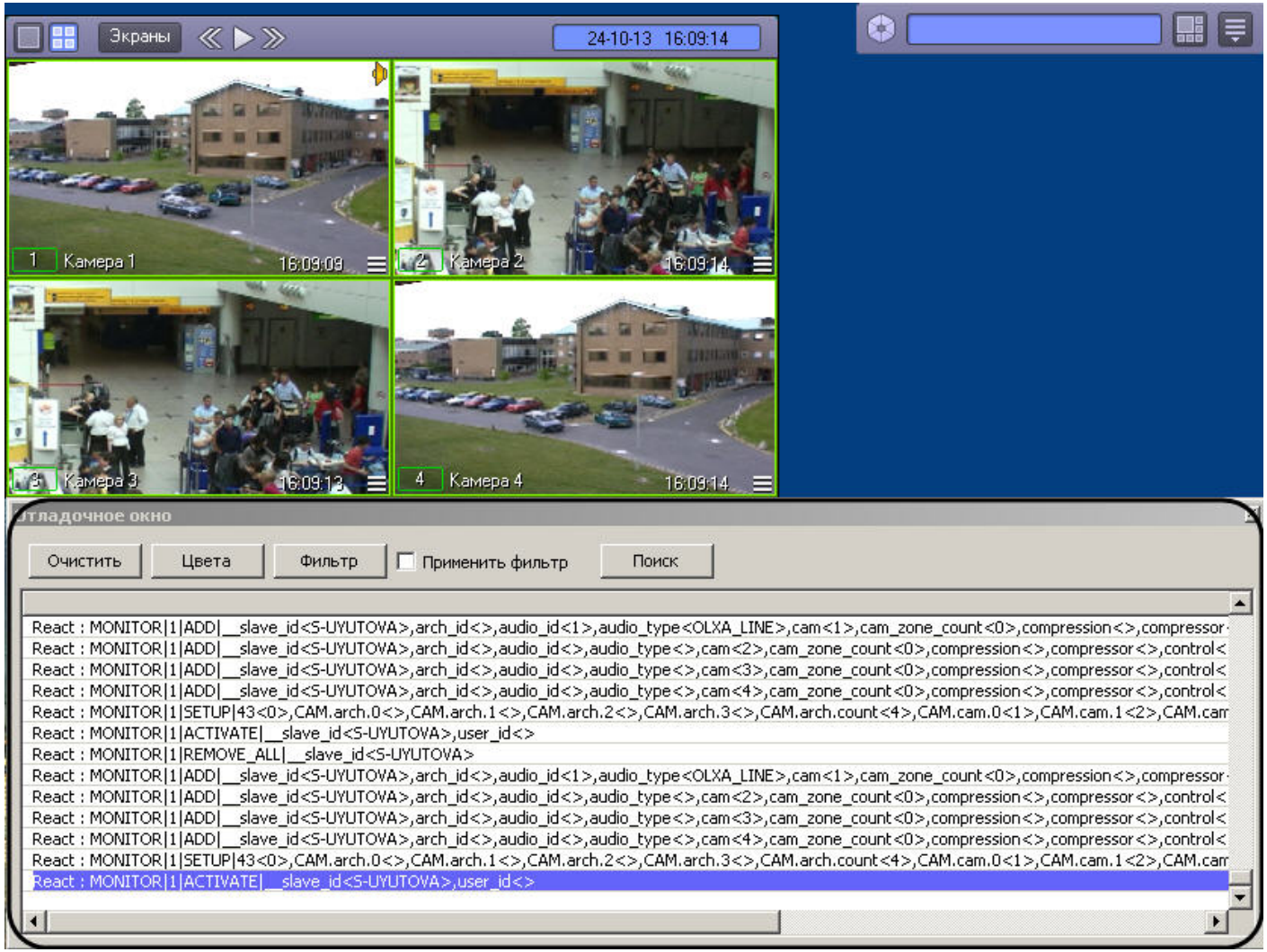


Для того чтобы скрыть Отладочное окно, требуется повторно выбрать пункт **Отладочное окно** в Главной панели управления.

Примечание.
Для выключения Отладочного окна следует выбрать значение **Выключен** для параметра **Режим отладки** в утилите tweaki.exe, либо установить строковый параметр «Debug» равным 0 в разделе «HKEY_LOCAL_MACHINE\ SOFTWARE\ ITV\ Intellect» реестра ОС Windows (HKEY_LOCAL_MACHINE \Software\Wow6432Node\ITV\Intellect для 64-битной системы). Данные операции проводятся при выгруженном ПК *Интеллект*.

Работа с Отладочным окном

Внешний вид Отладочного окна представлен на рисунке. В Отладочном окне выводится последовательность событий и реакций в системе.



Отладочное окно обладает следующими свойствами:

1. располагается поверх других окон;
2. для изменения размеров Отладочного окна следует использовать мышь;
3. существует возможность копировать информацию о событии или реакции в буфер обмена Windows для последующего использования в программах;
4. существует возможность фильтровать события или реакции, отображаемые в отладочном окне;
5. существует возможность выделять цветом события или реакции, отображаемые в Отладочном окне;
6. существует возможность поиска событий или реакций в Отладочном окне.

При выделении цветом и фильтрации сообщений отладочного окна могут быть использованы регулярные выражения.

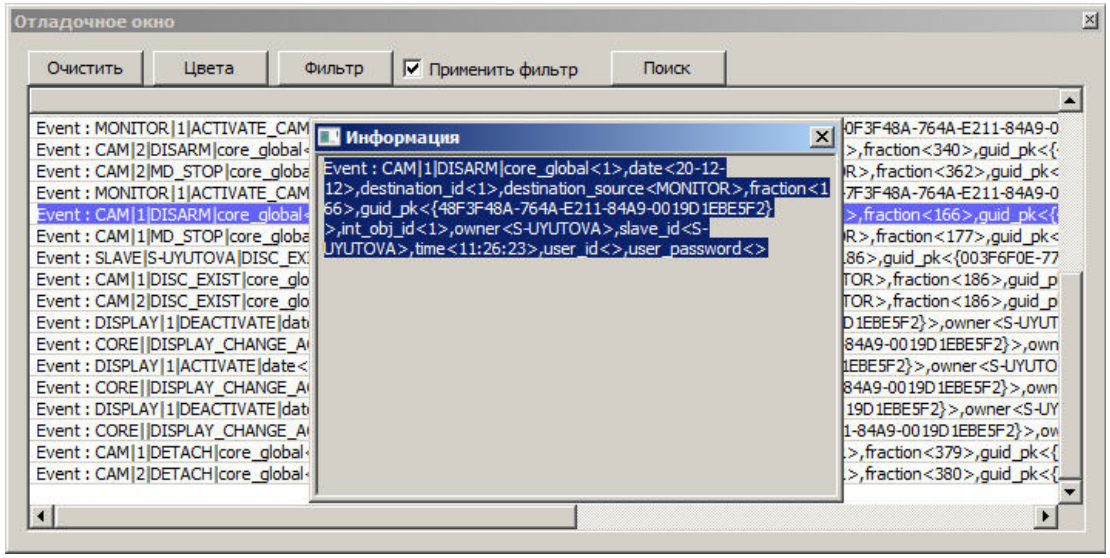
Копирование информации о событии или реакции в буфер обмена

Чтобы прочитать и/или скопировать в буфер обмена Windows информацию о событии или реакции, необходимо выполнить следующую последовательность действий:

1. Выделить требуемую строку в **Отладочном окне**.
2. Щелкнуть правой кнопкой мыши по выделенной строке. В результате выполнения операции появится окно **Информация**, содержащее информацию о требуемом событии или реакции.
3. Для копирования в буфер обмена Windows следует выделить требуемые сведения, после чего нажать **Ctrl**+**C**.

> **Примечание.**
> Для операций с текстом в окне **Информация** удобно использовать контекстное меню (вызывается щелчком правой кнопкой мыши по выделенному тексту).

4. Для закрытия окна **Информация** необходимо нажать  .

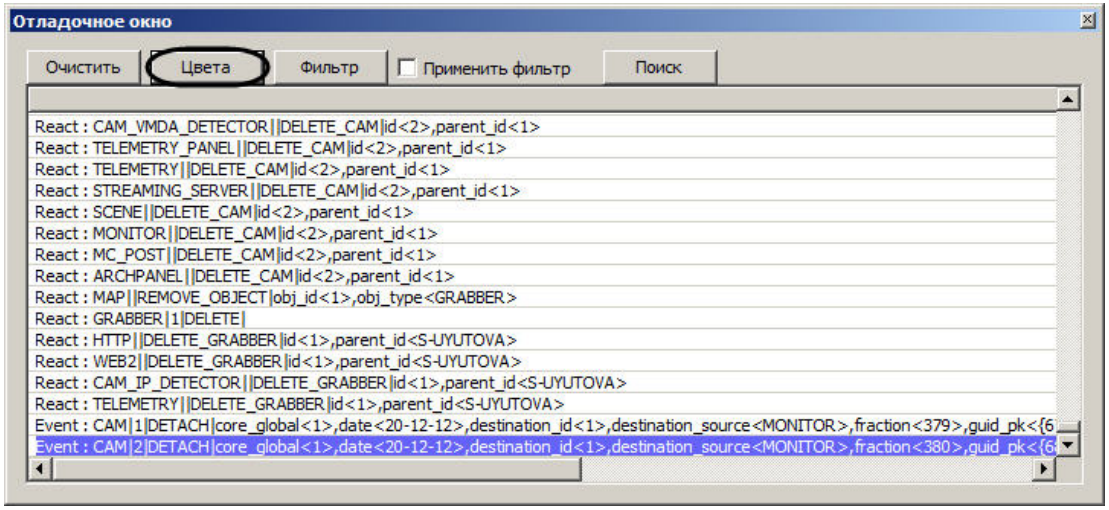


Копирование информации о событии или реакции в буфер обмена Windows завершено.

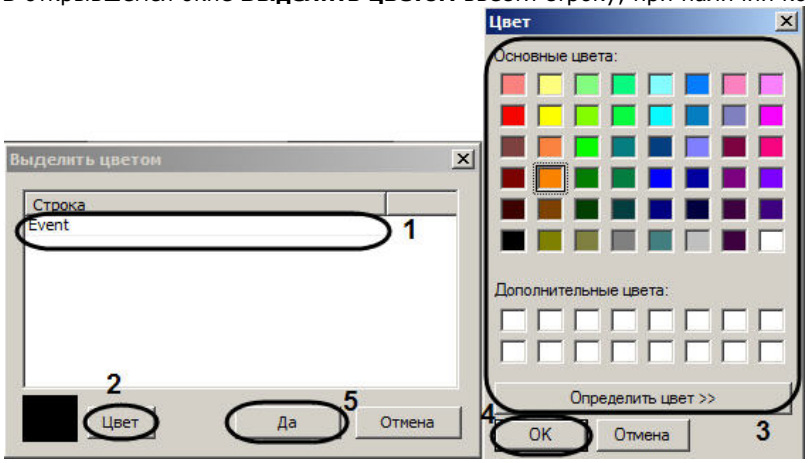
Выделение сообщений цветом

Для того, чтобы настроить выделение сообщений в отладочном окне цветом, необходимо выполнить следующие действия:

1. Нажать на кнопку **Цвета**.



2. В открывшемся окне **Выделить цветом** ввести строку, при наличии которой в сообщении оно должно быть выделено цветом (1).

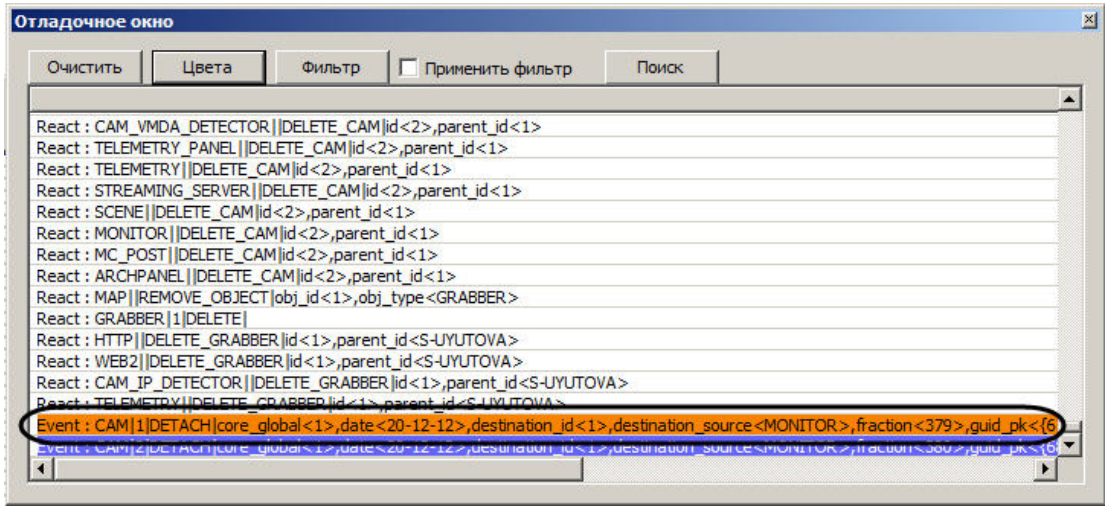


3. Нажать на кнопку **Цвет** (2).
4. В стандартном диалоговом окне Windows **Цвет** выбрать цвет подсветки сообщения (3).
5. Нажать на кнопку **ОК** (4).
6. Повторить шаги 2-5 для всех требуемых строк.

> **Примечание.**
> Для добавления строки в таблицу следует нажать на клавишу "вниз" на клавиатуре.

7. Нажать на кнопку **Да** (5).

В результате выполнения указанных действий в Отладочном окне будут подсвечены выбранным цветом сообщения, включающие введенную строку.



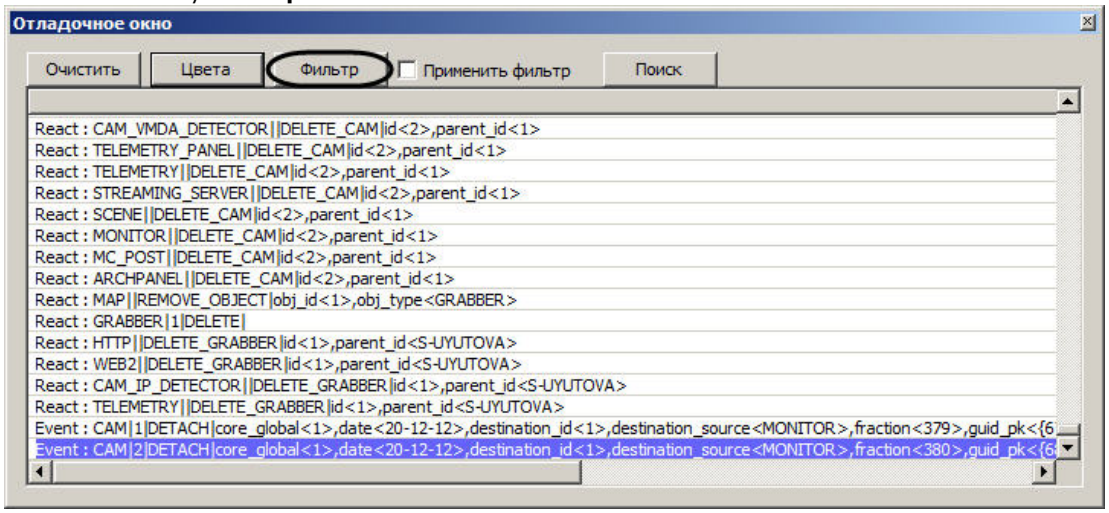
Выделение сообщений цветом завершено.

Фильтр событий и реакций

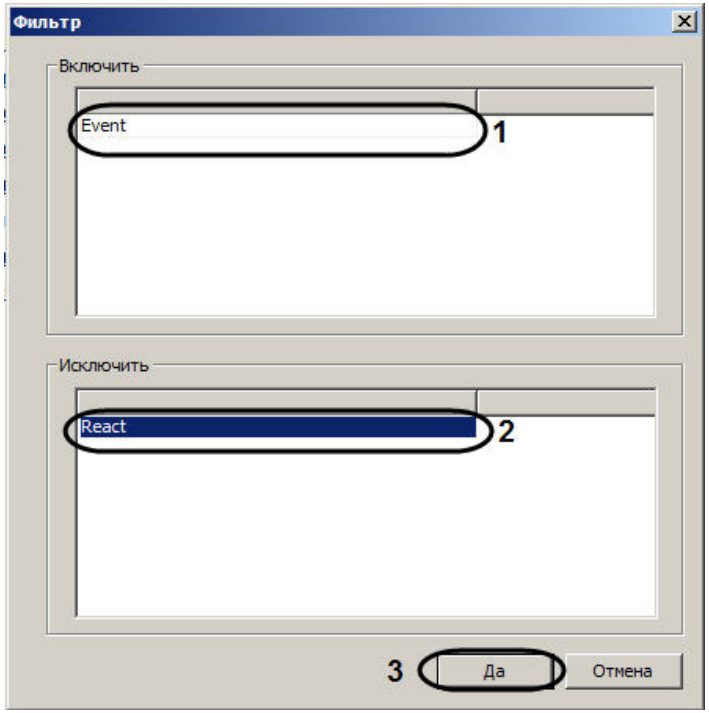
Фильтр событий и реакций позволяет отображать в Отладочном окне только требуемые сообщения.

Для настройки фильтра событий и реакций необходимо выполнить следующие действия:

1. Нажать на кнопку **Фильтр.**



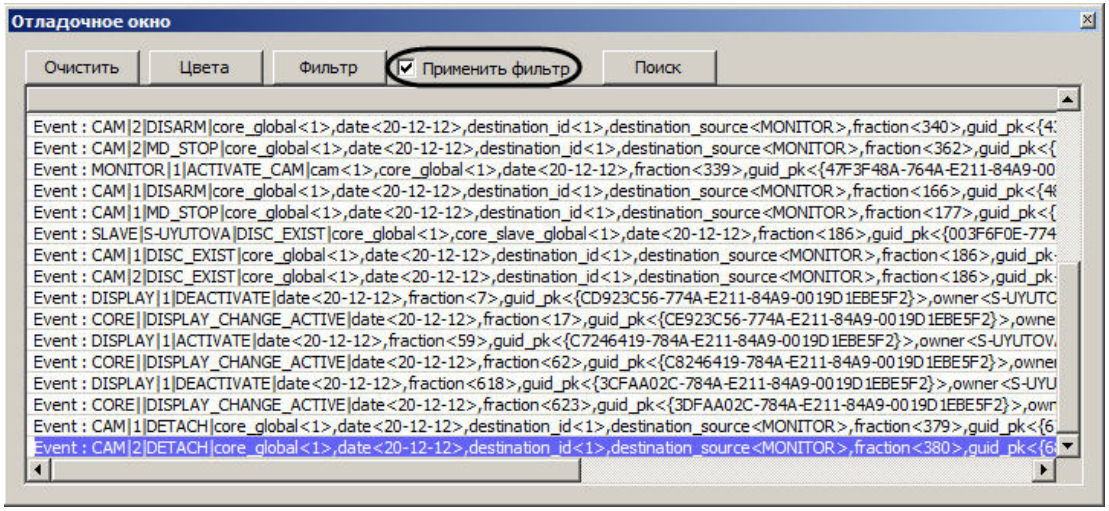
2. В открывшемся окне **Фильтр** указать строки, которые должны содержаться в сообщении, чтобы оно было отображено в Отладочном окне (1).



3. Указать строки, при наличии которых в сообщении оно не отображается в Отладочном окне (2).

> **Примечание.**
> Для добавления строки в таблицу следует нажать на кнопку "вниз" на клавиатуре.

4. Нажать на кнопку **Да** (3).
5. Для применения фильтра установить флажок **Применить фильтр.**
   В результате в отладочном окне будут отображены только сообщения, удовлетворяющие условиям фильтра.

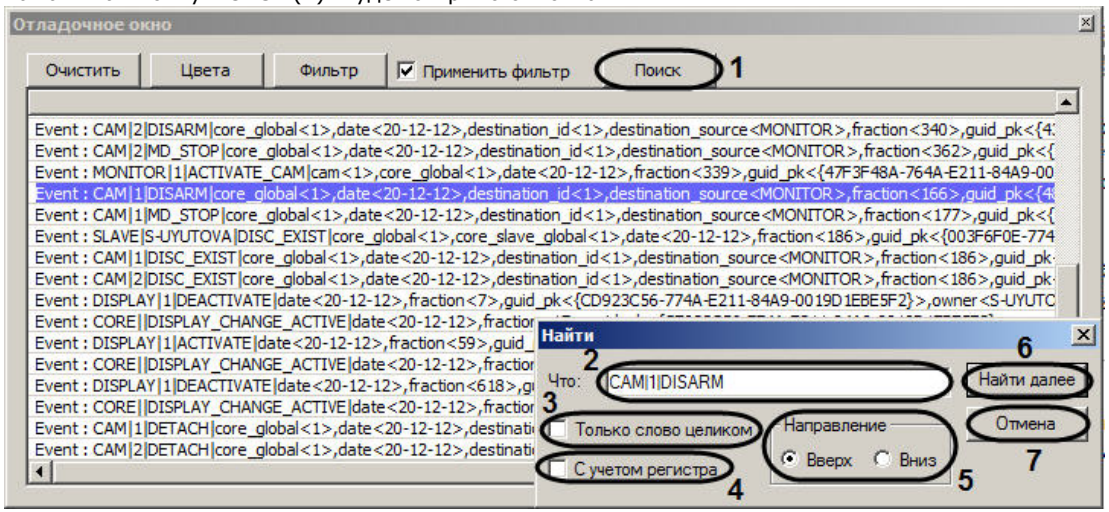


Настройка фильтра событий и реакций завершена.

Поиск событий и реакций

Поиск событий и реакций осуществляется следующим образом:

1. Нажать на кнопку **Поиск** (1). Будет открыто окно **Найти**.



2. Ввести в поле **Что:** условие поиска (2).
3. В случае, если требуется искать введенную строку как самостоятельное слово, не присутствующее в других словах в виде части, а отделенное от них как минимум одним пробелом, необходимо установить флажок **Только слово целиком** (3).
4. В случае, если при поиске следует учитывать регистр символов, необходимо установить флажок **С учетом регистра** (4).
5. Установить переключатель **Направление** в положение, соответствующее направлению поиска (5).

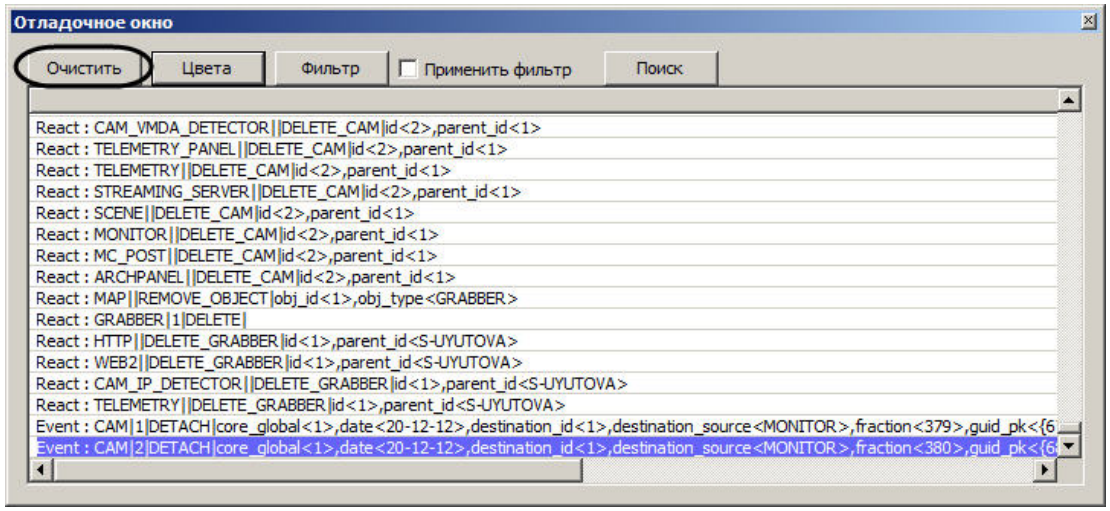
6. Для просмотра следующего результата поиска нажать на кнопку **Найти далее** (6).

> **Примечание.**
> Для закрытия окна **Найти** нажать на кнопку **Отмена.**

Поиск событий и реакций завершен.

Очистка Отладочного окна

Для того, чтобы удалить из Отладочного окна все сообщения необходимо нажать на кнопку **Очистить**.



Получение списка системных названий объектов, реакций и событий ПК Интеллект

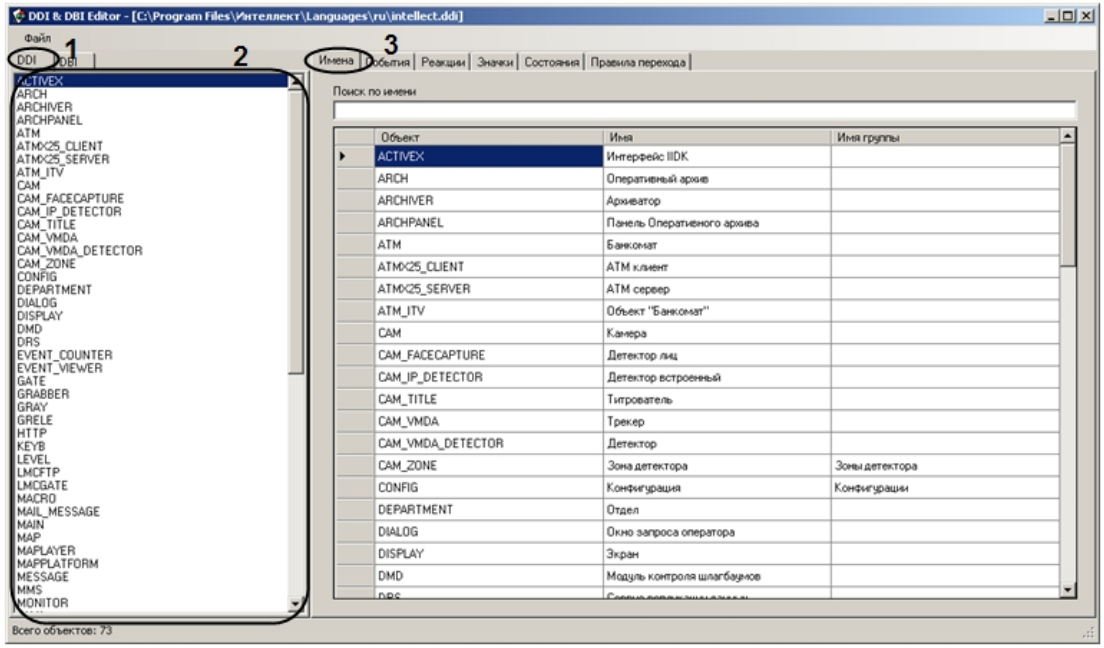
Список системных названий объектов, реакций и событий программного комплекса *Интеллект*, используемых при программировании, можно получить при помощи утилиты настройки конфигурации *ddi.exe.*

Утилита *ddi.exe* запускается одним из следующих способов:

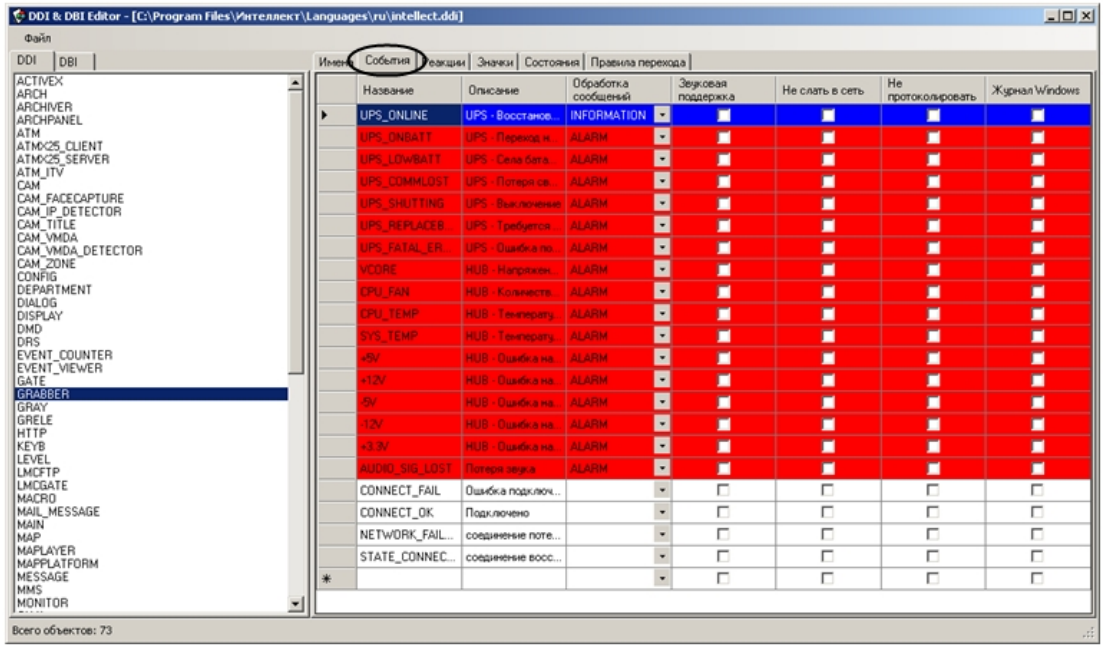
1. Из меню **Пуск** -> **Программы** -> **Интеллект** -> **Утилиты** -> **Настройка конфигурации**.
2. Из папки **Tools** директории установки ПК *Интеллект*.

Для просмотра списка системных названий объектов, событий и реакций необходимо выполнить следующие действия:

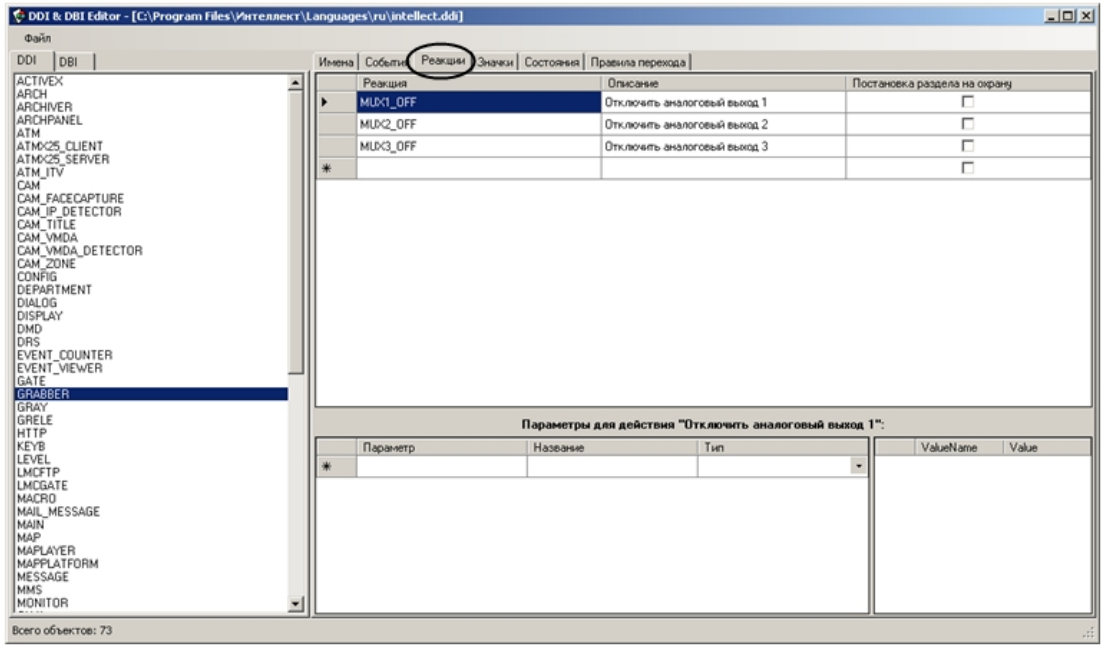
1. Открыть в утилите файл intellect.ddi.
2. Выбрать вкладку **DDI** в левой части окна утилиты (1). В списке на данной вкладке представлен перечень системных объектов.



3. Выбрать в списке на вкладке **DDI** объект, события и реакции которого требуется просмотреть (2).
4. Для просмотра имени выбранного объекта необходимо перейти на вкладку **Имена** (3).
5. Для просмотра списка событий выбранного объекта необходимо перейти на вкладку **События** .



6. Для просмотра списка реакций выбранного объекта необходимо перейти на вкладку **Реакции**.



Более подробно работа с утилитой ddi.exe описана в документе Руководство администратора.

> **Примечание.**
> Если луч поставлен на охрану, то при замыкании/размыкании луча, в зависимости от настройки режима срабатывания приходит событие "Тревога" (см. Руководство по установке и настройке компонентов охранной системы, раздел Создание и настройка системного объекта Луч). Если луч снят с охраны, то приходят события "Замкнут"/"Разомкнут" соответственно.

Создание, сохранение и удаление скрипта

Создание скрипта

Создание скриптов на языке JScript в программном комплексе *Интеллект* осуществляется с использованием встроенной утилиты *Редактор – Отладчик*.

Утилита *Редактор-Отладчик* запускается с помощью кнопки **Редактор-Отладчик**, расположенной на панели настройки системного объекта **Скрипт.**

Для создания скрипта на языке программирования JScript в программном комплексе *Интеллект* необходимо выполнить следующие действия:

1. На вкладке **Программирование** диалогового окна **Настройка системы** создать объект **Скрипт**. Задать объекту **Скрипт** идентификационный номер и название.
2. В поле **Временная зона** требуется указать временную зону выполнения скрипта (например, зону **Всегда).**

   > **Примечание.**
   > По умолчанию для выполнения скрипта указана временная зона **Никогда**.

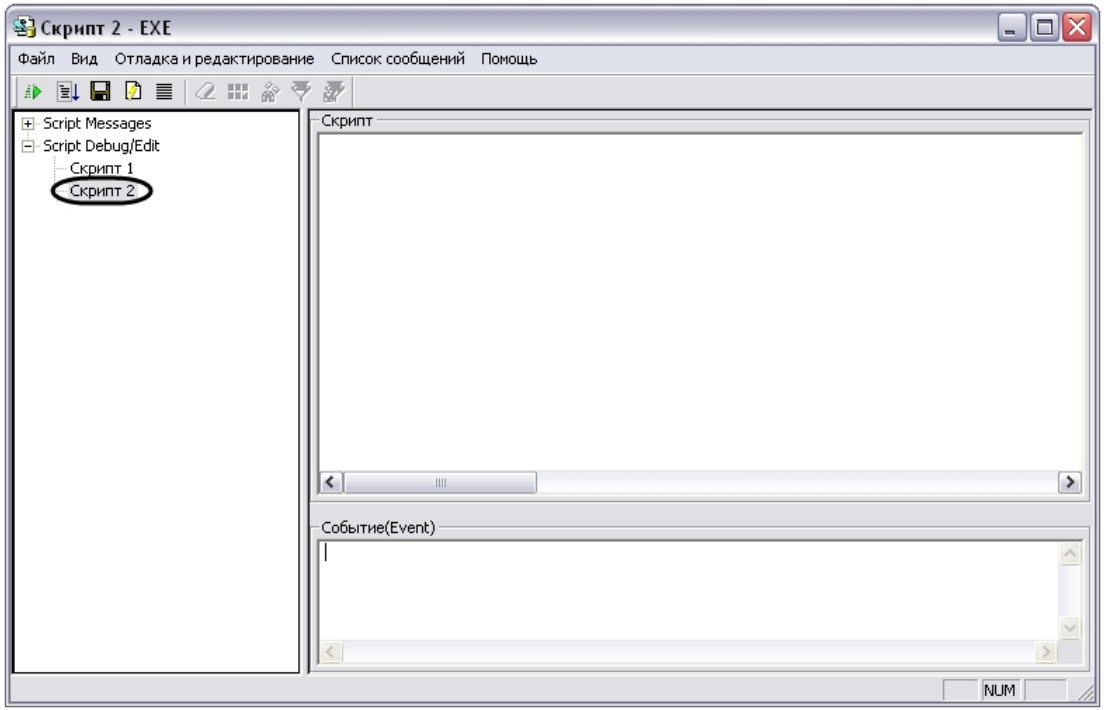
3. В панели **Компьютеры** указать компьютеры (ядра), на которых требуется выполнять создаваемый скрипт.

> **Примечание.**
> По умолчанию скрипт настроен на выполнение на всех компьютерах (ядрах). В списке компьютеров отображаются только те компьютеры, которые зарегистрированы на вкладке **Оборудование** диалогового окна **Настройка системы**.

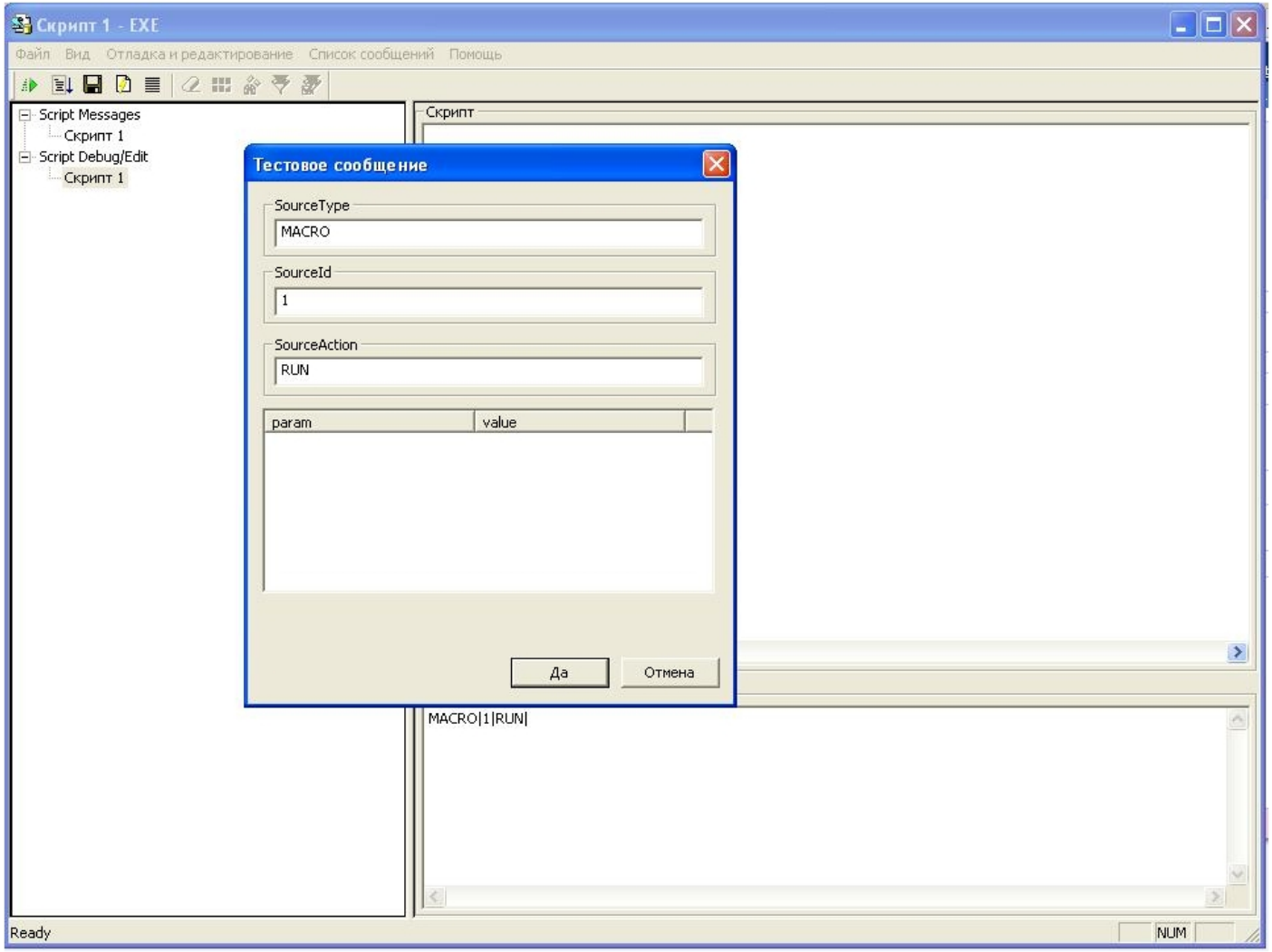
4. С помощью кнопки **Редактор-Отладчик**, расположенной в нижней части панели системного объекта **Скрипт**, вызвать утилиту *Редактор-Отладчик*.

> **Примечание.**
> Создание, редактирование и сохранение скрипта на языке программирования JScript рекомендуется осуществлять с помощью утилиты *Редактор-Отладчик*. На панели настройки системного объекта **Скрипт** отображается текст сохраненного скрипта, также доступный для редактирования.

5. В окне утилиты *Редактор-Отладчик* необходимо раскрыть список **Script Debug\Edit** и выбрать объект **Скрипт**, редактирование которого необходимо выполнить (например, объект **Скрипт 2**, как на рисунке ниже).



6. В поле **Скрипт** требуется ввести текст скрипта на языке программирования JScript.
7. Запустить скрипт, воспользовавшись тестовым событием. Для создания тестового события воспользоваться командой **Отладка и редактирование -> Редактировать тестовое событие.** В результате выполнения данной команды на экран будет выведено окно **Тестовое сообщение**, содержащее поля для задания параметров события.



8. Для запуска скрипта по тестовому событию необходимо воспользоваться командой **Отладка и редактирование -> Тестовый пуск**.
9. Проверить скрипт на корректность синтаксиса. Данная проверка выполняется встроенным в утилиту *Редактор-Отладчик* интерпретатором. Результат проверки с информацией о содержании и местонахождении ошибки отображается в соответствующем скрипту **Отладочном окне** в списке **Script Messages**. При наличии ошибок необходимо внести правки в синтаксис скрипта и повторить проверку.

> **Примечание.**
> Подробная информация об использовании тестовых событий для отладки скриптов приведена в главе Отладка скриптов.

10. После отладки скрипта средствами утилиты *Редактор-Отладчик* запустить его по реальному системному событию. Проверить результат выполнения скрипта. В случае некорректности выполнения скрипта, внести необходимые изменения и повторно запустить скрипт.

Процесс создания скрипта можно считается завершенным в том случае, если скрипт выполняется корректно.

Смотреть видео

Сохранение скрипта

Утилита *Редатор-Отладчик* обеспечивает два способа сохранения скриптов: в системном объекте **Скрипт** или в текстовом файле на диске компьютера.

Сохранение скрипта в системном объекте **Скрипт** осуществляется по команде **Файл -> Сохранить в базе**.

Примечание.
Скрипт автоматически сохраняется в соответствующем ему системном объекте **Скрипт** при завершении работы с утилитой *Редактор-Отладчик*.

Сохранение скрипта в файл выполняется по команде **Файл -> Сохранить на диск**. Сохраненный в файл скрипт впоследствии может быть загружен в утилиту *Редактор-Отладчик* с помощью команды **Файл -> Загрузить с диска**.

Смотреть видео

Удаление скрипта

Удаление созданного в программном комплексе *Интеллект* скрипта осуществляется путем удаления соответствующего ему системного объекта **Скрипт**, размещенного во вкладке **Программирование**.

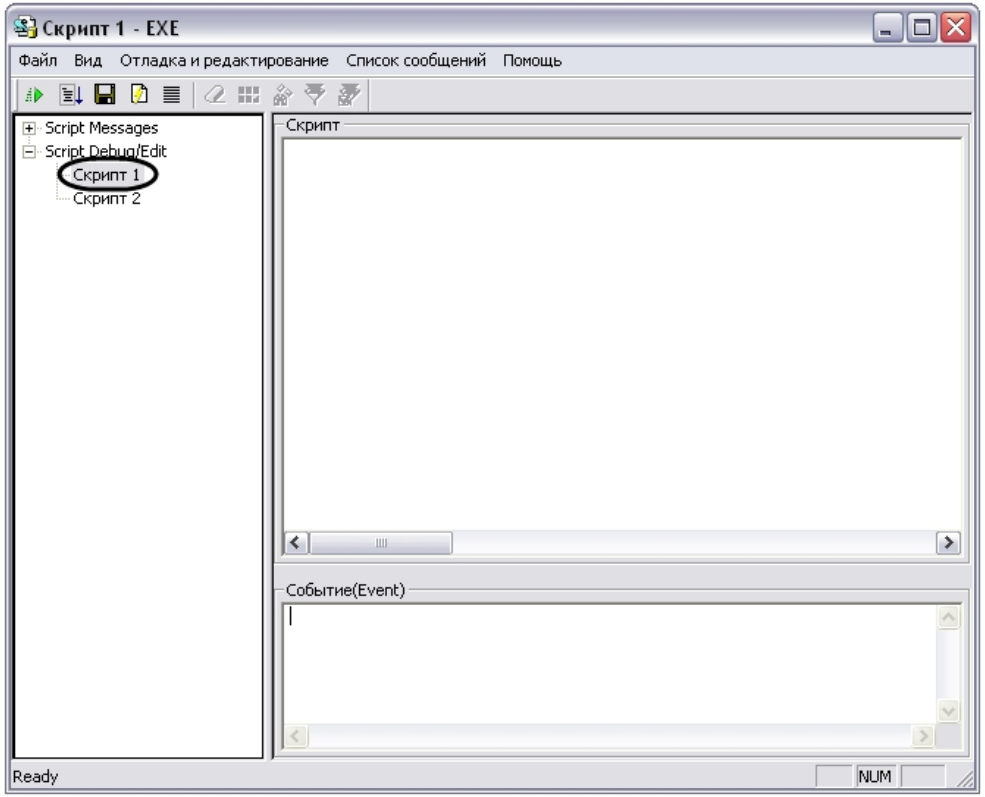
Смотреть видео

Создание первого скрипта

В качестве примера использования языка программирования JScript в программном комплексе *Интеллект,* вначале предлагается создать скрипт, содержащий ошибку, а впоследствии внести в него исправления. Скрипт выполняет следующие действия: по запуску **Макрокоманды № 1** скрипт должен устанавливать для камер № 1 – 4 значение параметра **Горячая запись** равным 10 и выводит в отладочное окно утилиты *Редактор-Отладчик* сообщение "Hello world".

Для создания и запуска данного скрипта необходимо выполнить следующие действия:

1. Во вкладке **Оборудование** диалогового окна **Настройка системы** создать четыре объекта **Камера** с идентификационными номерами 1, 2, 3 и 4, если они не были созданы ранее.
2. Во вкладке **Программирование** создать объект **Макрокоманда** с идентификационным номером 1. Таблицу **События** заполнять не требуется для корректного выполнения последующих действий и успешного запуска скрипта.
3. Во вкладке **Программирование** создать системный объект **Скрипт.** Задать объекту идентификационный номер 1 и название "Скрипт 1".
4. В панели настройки системного объекта **Скрипт 1** из списка **Временная зона** выбрать пункт **Всегда.**
5. Нажать кнопку **Редактор-Отладчик**, расположенную в нижней части панели настройки системного объекта **Скрипт 1**. После выполнения указанного действия на экран будет выведено окно утилиты *Редактор-Отладчик*.
6. В окне утилиты *Редактор-Отладчик* необходимо раскрыть список **Script Debug\Edit** и выбрать объект **Скрипт 1**.

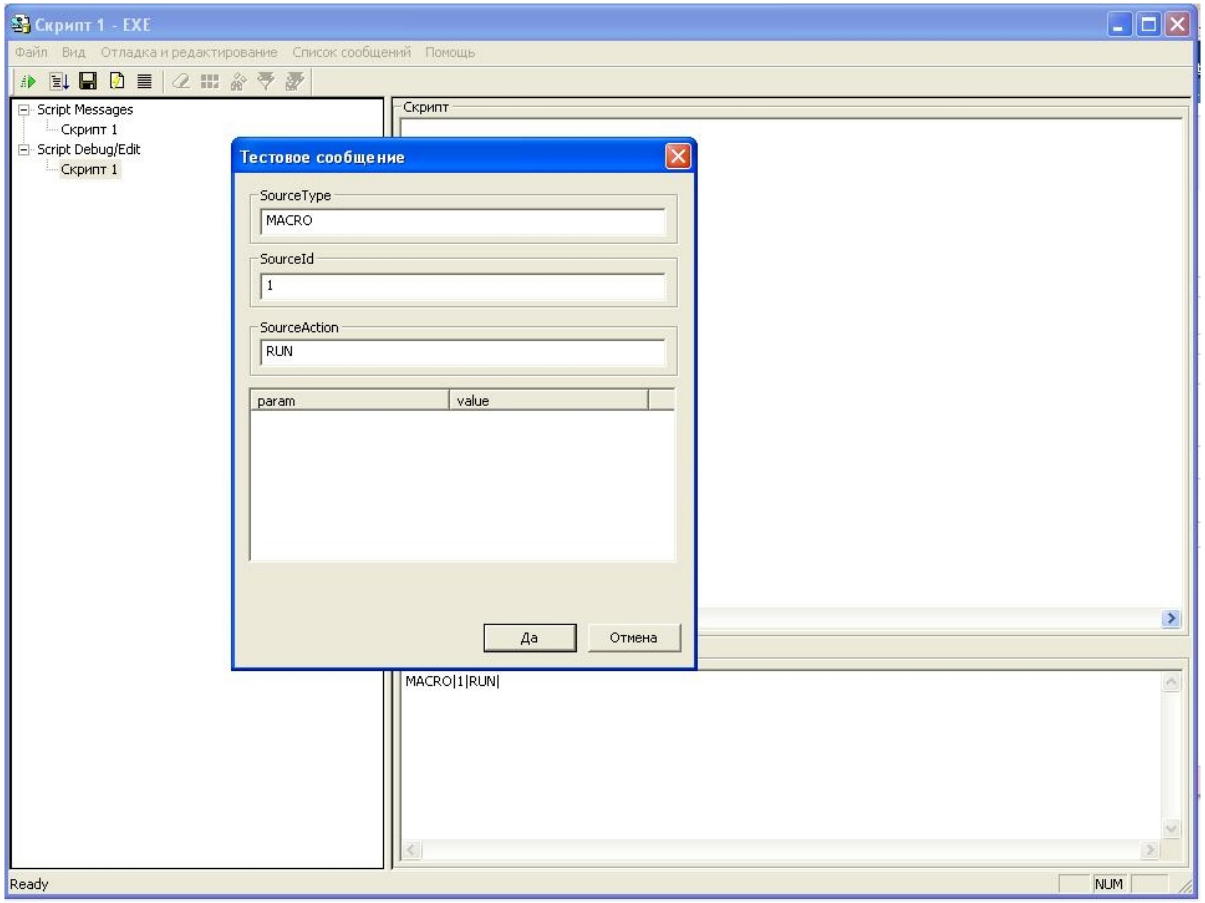


7. В поле **Скрипт** ввести следующие строки:

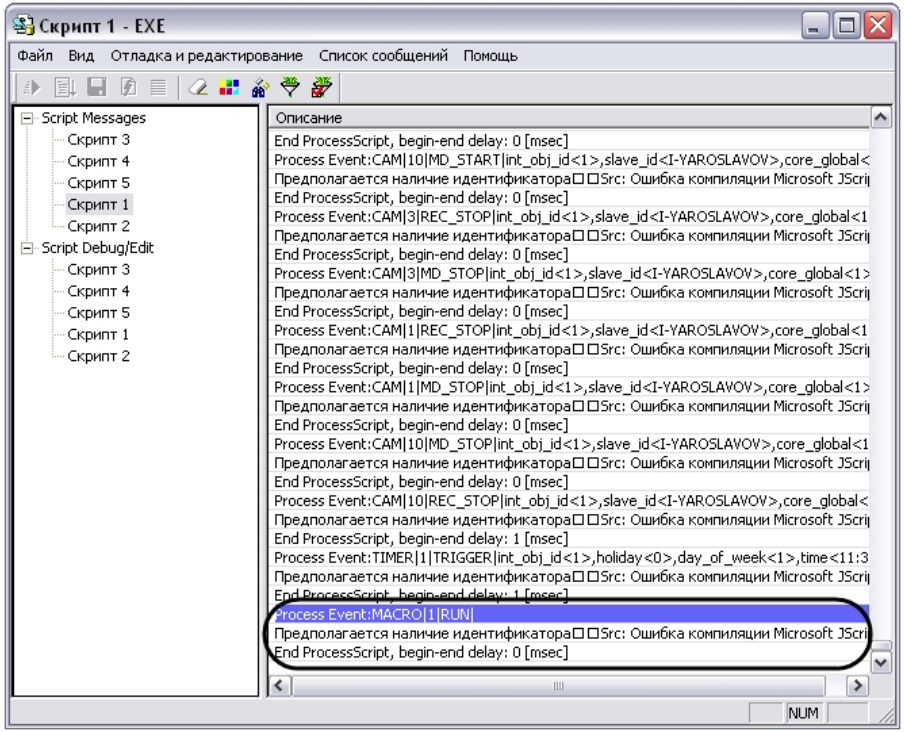
```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
 var ;
 for(i=1;i<=4;i=i+1)
 {
  SetObjectParam("CAM",i,"hot_rec_time","10");
 }
 DebugLogString ("Hello world");
}
```

Внимание!
Скрипт содержит ошибку. Рекомендации по ее устранению приводятся ниже.

8. Сохранить скрипт, выбрав в меню **Файл** утилиты команду **Сохранить в базе**.
9. Создать тестовое событие для запуска скрипта в режиме отладки – MACRO|1|RUN|. Для этого необходимо выбрать в меню **Отладка и редактирование** команду **Редактировать тестовое событие**, при этом на экран будет выведено окно **Тестовое сообщение**. Необходимо заполнить поля окна **Тестовое сообщение**, как показано на рисунке.



10. Запустить скрипт по тестовому событию, выбрав в меню **Отладка и редактирование** команду **Тестовый Пуск**.
11. Раскрыть список **Script Messages** и выбрать пункт **Скрипт 1**. В правой части окна утилиты отобразится **Отладочное окно** скрипта.
12. В отладочном окне найти строку Process Event:MACRO|1|RUN| и сообщение об ошибке: "Предполагается наличие идентификатора Src: Ошибка компиляции Microsoft JScript Line:2 Char:6 Error:0 Scode:800a03f2 ".

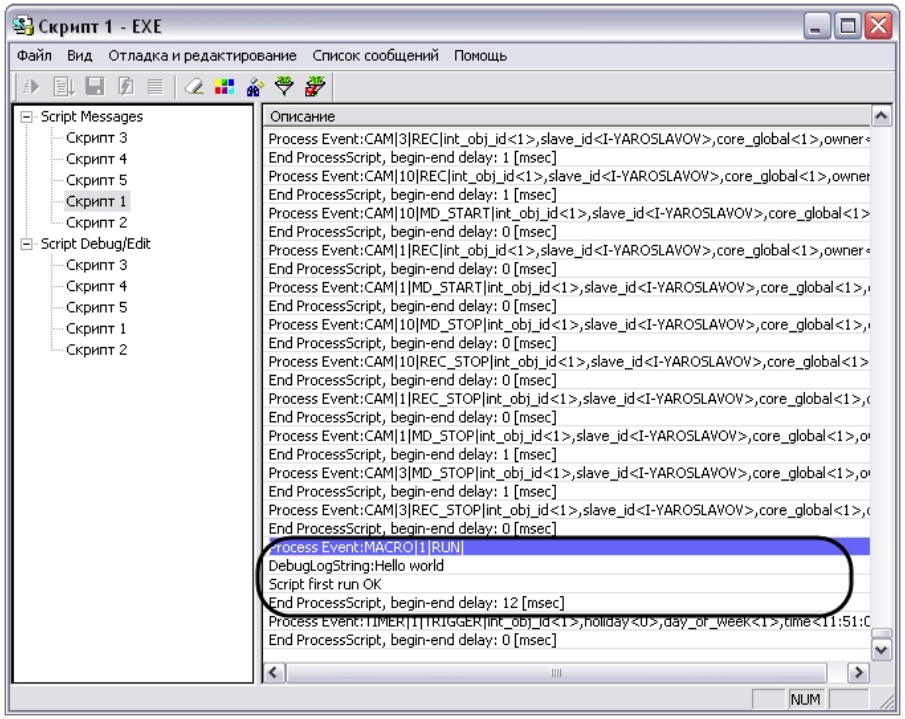


Сообщение об ошибке указывает, что в строке 2 данного скрипта в операторе объявления переменных (var) отсутствует идентификатор, то есть ни одна переменная объявлена не была. В соответствии с правилами языка JScript, это считается критической ошибкой, и выполнение скрипта не осуществляется.

13. Внести исправления в текст скрипта, как показано ниже (см. строку var i;).

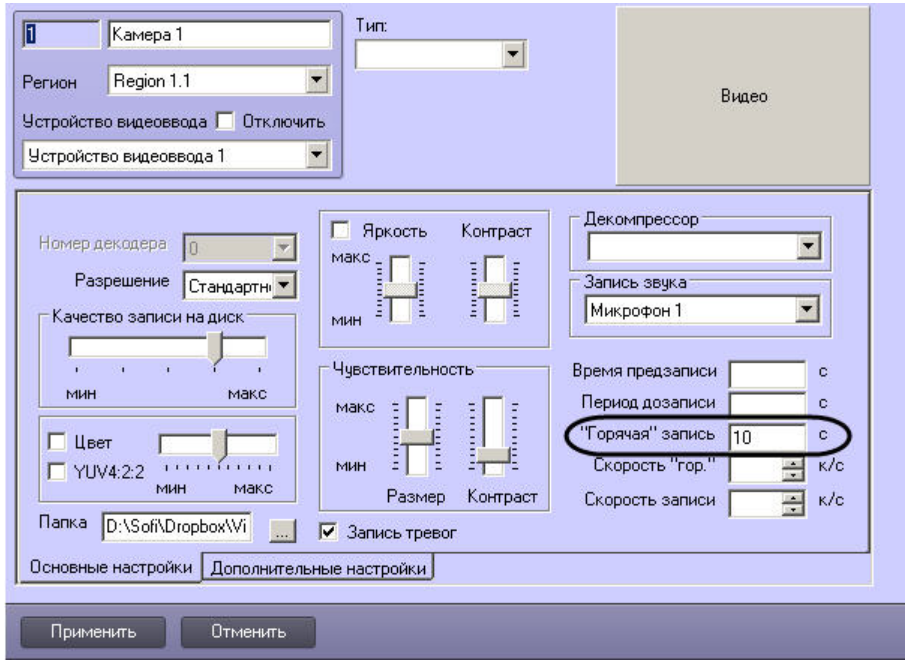
```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
 var i;
 for(i=1; i<=4; i=i+1)
 {
  SetObjectParam("CAM",i,"hot_rec_time","10");
 }
 DebugLogString ("Hello world");
}
```

14. Сохранить скрипт, выбрав в меню **Файл** утилиты команду **Сохранить в базе**.
15. Повторить действия 10 и 11.
16. В отладочном окне найти строку "Process Event:MACRO|1|RUN|" и сообщения" DebugLogString:Hello world" и "Script first run OK". Сообщение "Script first run OK" свидетельствует о том, что скрипт корректно работает в режиме отладки.



17. Завершить работу с утилитой *Редактор-Отладчик.*
18. В поле системного объекта **Скрипт 1** отобразится текст созданного скрипта. Для активирования скрипта в панели настройки системного объекта **Скрипт 1** требуется нажать кнопку **Применить**.
19. Вызвать из меню **Выполнить** Главной панели управления макрокоманду №1.
20. С помощью отладочного окна ПК *Интеллект* убедиться в успешном запуске макрокоманды и выполнении скрипта .

21. Убедиться в корректном выполнении скрипта. В панели настойки системных объектов **Камера № 1-4** в поле **Горячая запись** должно быть указано значение 10.



Примечание.
По умолчанию поля **Горячая запись** в панелях настройки объектов **Камера** не заполнены.

Процесс создания и отладки скрипта завершен.

Отладка скриптов

Возможности отладки скриптов

Утилита *Редактор-Отладчик* обеспечивает отладку скриптов с помощью встроенных программных средств проверки корректности синтаксиса, интерпретации скриптов, запуска по тестовым событиям, генерируемым утилитой. Процесс отладки сопровождается отображением сообщений о результатах проверки и выполнения скрипта в отладочных окнах.

Утилита *Редактор-Отладчик* обеспечивает следующие возможности отладки скриптов:

1. Каждому системному объекту **Скрипт** соответствует отдельное отладочное окно, в которое выводятся системные и тестовые события, сообщения об ошибках и успешном выполнении скриптов, а также пользовательские информационные сообщения. Предусмотрено использование фильтров вывода сообщений в отладочных окнах.
2. Предусмотрено использование специализированных отладочных окон **Информационное окно**, в которых отображаются сообщения, относящиеся только к отлаживаемому скрипту.
3. Для проверки работоспособности скрипта могут быть использованы тестовые события, генерируемые утилитой *Редактор-Отладчик* (тестовые события не регистрируются в системе).
4. Допускается использование сторонних программ-отладчиков, реализующих функции пошагового выполнения скриптов (Step), просмотра значений переменных

скриптов в процессе их выполнения (Watch) и др.

Смотреть видео

Создание и использование тестовых событий

На странице:

- Создание тестовых событий
- Запуск скрипта по тестовому событию
- Смотреть видео

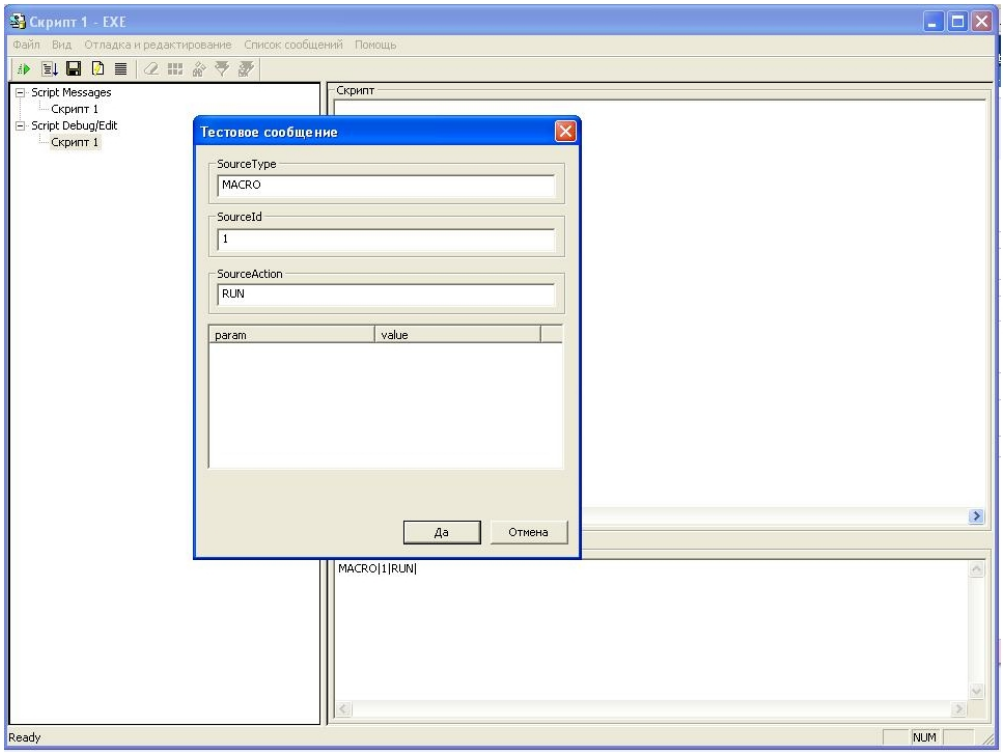
Создание тестовых событий

Для удобства отладки скриптов в утилите *Редактор-Отладчик* реализована возможность использования тестовых событий, задаваемых пользователем и генерируемых утилитой. Тестовые события не регистрируются на уровне системы видеонаблюдения: не отображаются в протоколе событий и не записываются в базу данных.

Для каждого скрипта допускается создание не более одного тестового события.

Для создания тестового события требуется выполнить следующие действия:

1. Выбрать в меню **Отладка и редактирование** команду **Редактировать тестовое событие**. Данная команда также может быть вызвана с панели инструментов нажатием кнопки .
2. На экран будет выведено окно **Тестовое сообщение**. Данное окно предназначено для ввода параметров тестового события.

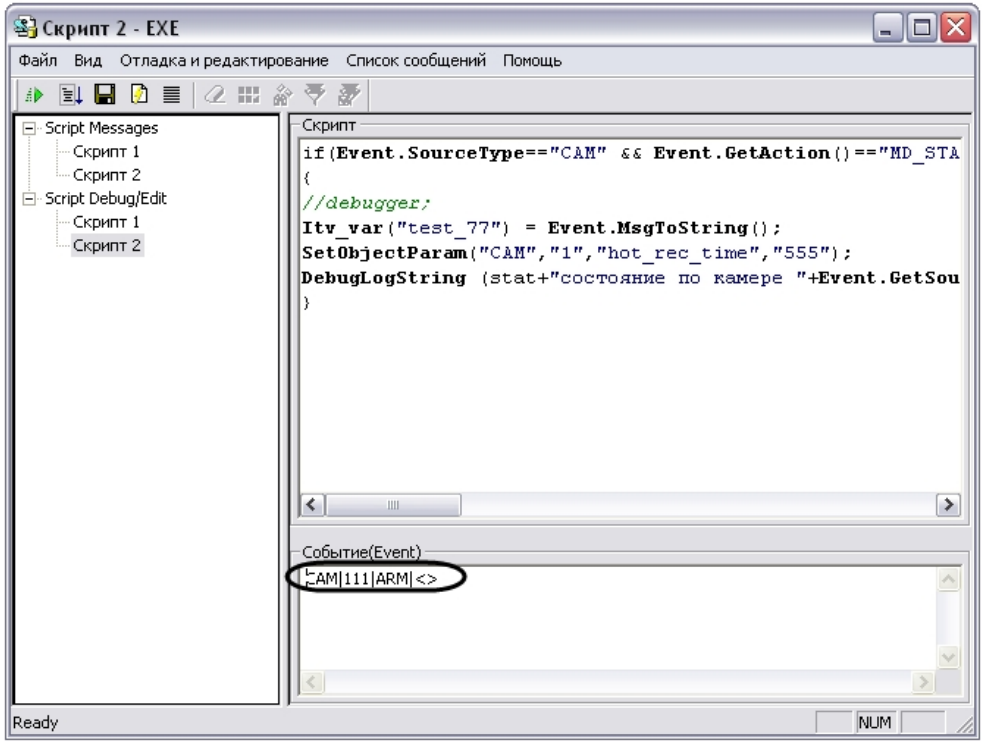


3. В полях окнах **Тестовое сообщение** требуется ввести следующую информацию:
    a. **SourceType** – тип системного объекта;
    b. **SourseId** – идентификационный номер системного объекта;
    c. **SourceAction** – событие, генерируемое заданным системным объектом;
    d. **param** – дополнительные параметры события;
    e. **value** – значения дополнительных параметров.
4. По окончании заполнения полей окна **Тестовое сообщение** необходимо нажать кнопку **Да.**

Процесс создания тестового события завершен.

После создания тестовое событие отобразится в поле **Событие(Event)** в специализированном строковом формате.

Например, на рисунке ниже в качестве тестового события указано **Постановка на охрану камеры № 111**.



Запуск скрипта по тестовому событию

Запуск скрипта по тестовому событию осуществляется одним из указанных способов:

1. нажать кнопку **Тестовый запуск** , расположенную на панели инструментов утилиты;
2. выбрать в меню **Отладка и редактирование** команду **Тестовый Пуск**;
3. выбрать в меню **Отладка и редактирование** команду **Тестовый пуск с Отладчиком**.

При запуске скрипта по команде **Тестовый пуск с Отладчиком** (см. п.3) осуществляется запуск сторонней программы-отладчика (детализированная информация приведена в разделе Использование сторонних программ-отладчиков).

Сообщения о результатах проверки и выполнения скрипта отображаются в соответствующем скрипту отладочном окне утилиты *Редактор-Отладчик*.

Смотреть видео

Работа с отладочными окнами утилиты Редактор-Отладчик

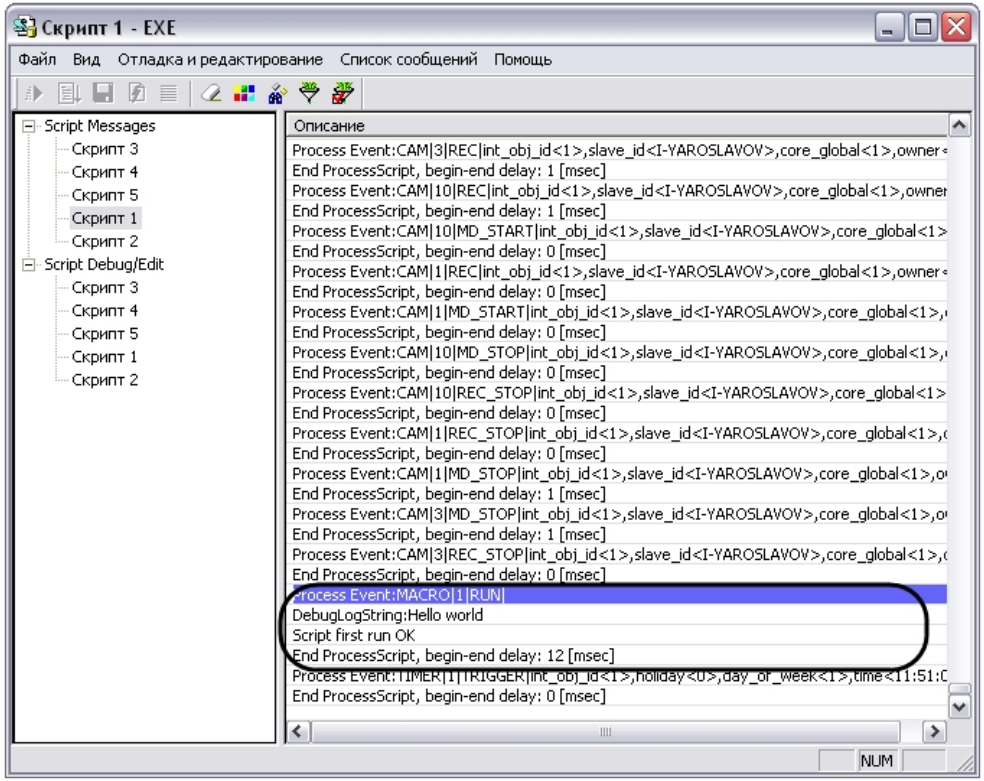
Типы отладочных окон: Script Messages и Информация от потока

Отладочные окна предназначены для отображения сообщений о регистрации системных и тестовых событий, об ошибках и успешном выполнении скриптов, а также пользовательских информационных сообщений.

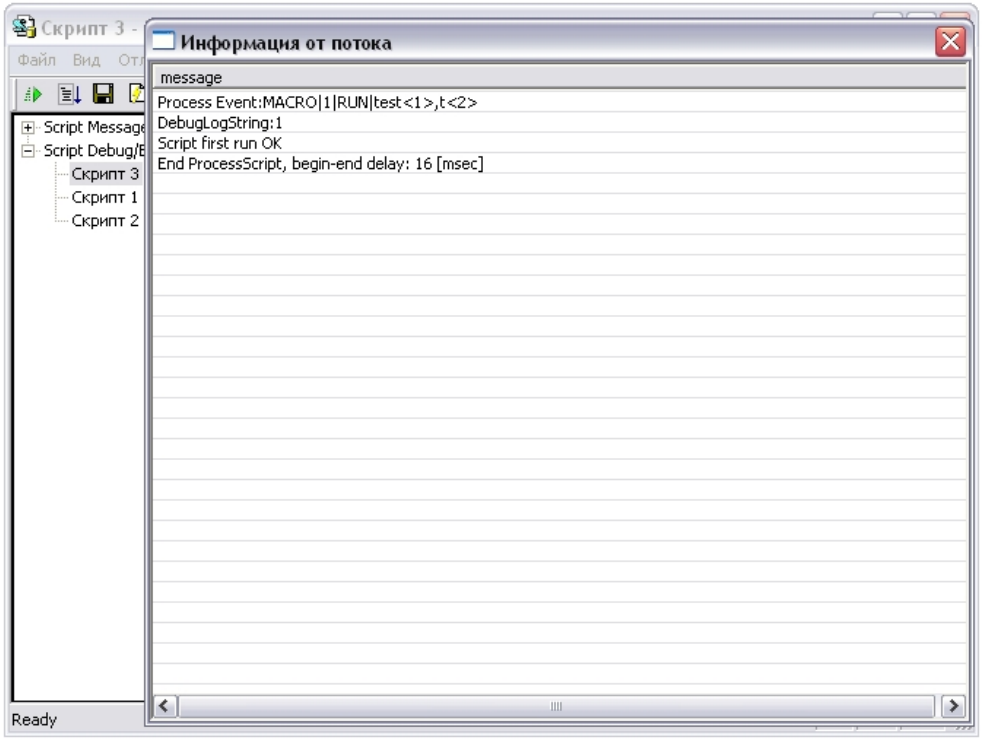
Для каждого скрипта в утилите *Редактор-Отладчик* предусмотрено отдельное отладочное окно.

Существует два типа отладочных окон: **Script Messages** и **Информация от потока**.

Отладочные окна типа **Script Messages** размещаются в списке **Script Messages**. Названия отладочных окон совпадают с названиями соответствующих им объектов **Скрипт**. В данные окна выводятся все системные сообщения, относящиеся ко всем зарегистрированным в программном комплексе *Интеллект* скриптам. Пример отладочного окна типа **Script Messages**:



Отладочные окна **Информация от потока** вызываются непосредственно из окон редактирования скриптов (окна списка **Script Debug/Edit**). Вызов окна осуществляется при активном окне редактирования скрипта по команде **Отладка и редактирование-> Сводная информация** или по нажатию кнопки  на панели инструментов. В окнах **Информация от потока** отображаются системные события, относящиеся только к отлаживаемому скрипту. Отладочное окно **Информация от потока:**



Порядок работы со всеми отладочными окнами любых типов одинаков.

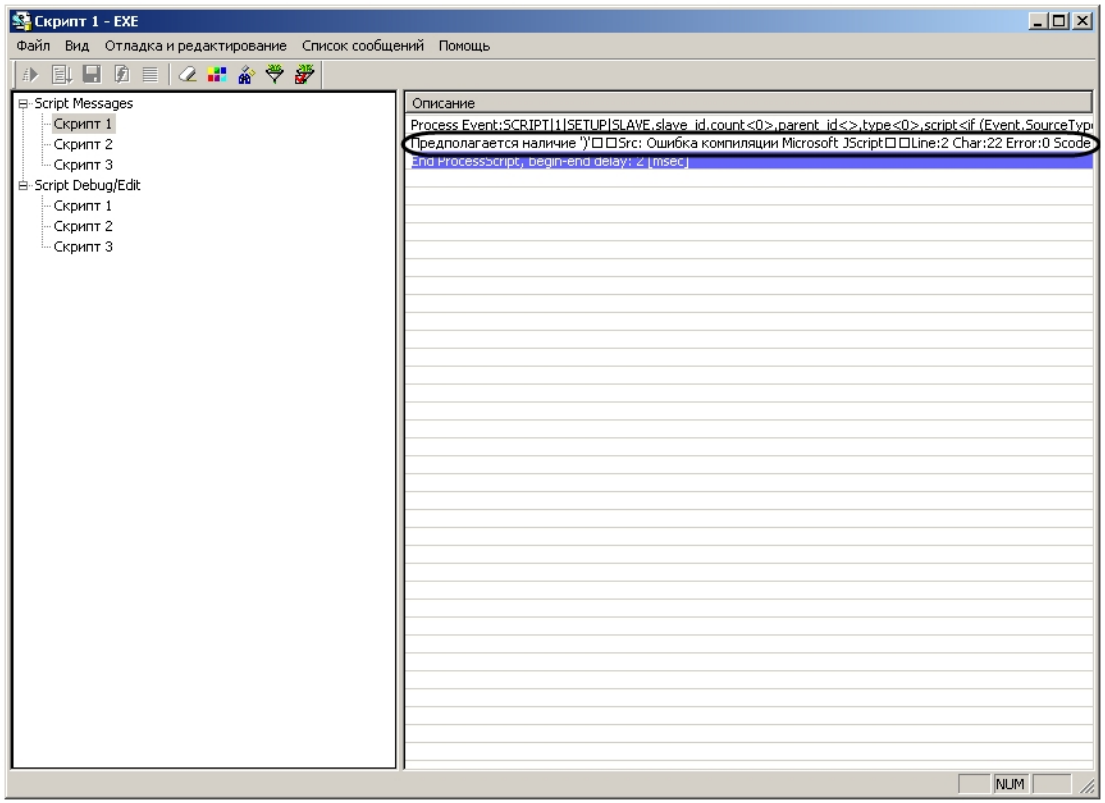
Смотреть видео

Отображение сообщений о запуске, проверке, изменении и выполнении скриптов в отладочных окнах

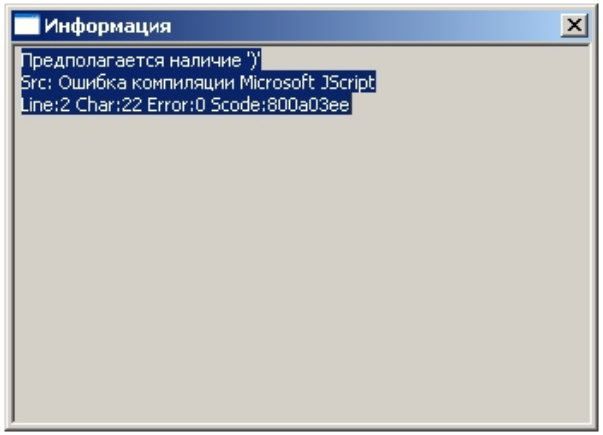
В отладочные окна последовательно выводятся сообщения, соответствующие этапам запуска, проверки и выполнения скриптов.

В момент регистрации события, по которому должен быть осуществлен запуск скрипта, в отладочное окно выводится сообщение «Process Event: событие, инициировавшее запуск скрипта» (например, при запуске скрипта по макрокоманде № 1 в отладочное окно выводится строка «Process Event:MACRO|1|RUN|»). В момент изменения скрипта в утилите *Редактор-Отладчик* или в ПК *Интеллект*, в отладочное окно выводится сообщение «Process Event:SCRIPT|номер скрипта|SETUP|» (например, при изменении скрипта с номером 1, в отладочное окно выводится строка «Process Event:SCRIPT|1|SETUP|»).

Перед запуском скрипта производится проверка корректности его синтаксиса. В том случае, если в процессе проверки синтаксиса скрипта были обнаружены ошибки, в отладочное окно будут выведены соответствующие сообщения. На рисунке приведен пример отображения в отладочном окне сообщения об ошибке при проверке синтаксиса скрипта.



Для просмотра полного содержания строки с сообщением требуется щелкнуть правой клавишей мыши по данной строке. В результате на экран будет выведено окно **Информация**, в котором содержащая сообщение об ошибке строка отображается полностью.



Строка с сообщением об ошибке содержит следующую информацию:

1. содержание ошибки;
2. тип ошибки (например, «Src: Ошибка компиляции Microsoft JScript»);
3. местоположение ошибки в тексте скрипта (номер строки «Line» и номер символа в строке «Char»);
4. код ошибки («Scode»).

В том случае, если при проверке синтаксиса скрипта не были обнаружены ошибки, в отладочное окно будет выведено сообщение: «Script first run OK». Далее будет осуществлен запуск скрипта.

В том, случае, если ошибки регистрируются в процессе выполнения скрипта, сообщения о них также выводятся в отладочное окне.

При успешном окончании выполнения скрипта в окно будет выведено сообщение «End ProcessScript, begin-end delay: время выполнения скрипта» (например, «End ProcessScript, begin-end delay: 13 [msec]»).

Смотреть видео

Использование сторонних программ-отладчиков

Программным комплексом *Интеллект* официально поддерживается отладчик Microsoft Visual Studio 2005.

ПК *Интеллект* допускает возможность использования сторонних программ для отладки скриптов на языке JScript. Сторонние программы-отладчики могут обеспечивать функциональные возможности отладки, не предусмотренные утилитой *Редактор-Отладчик*, например, пошаговое выполнение скриптов (функции "Step"), просмотр значений заданных в скриптах переменных в процессе выполнения скриптов (функция "Watch") и др.

Примечание.
Не рекомендуется использовать сторонние программы-отладчики, поскольку они не обеспечивают полную совместимость с программным комплексом *Интеллект*. Необходимо учитывать, что использование сторонних программ-отладчиков может привести к аварийному завершению работы ПК *Интеллект*.

В случае использования для отладки скриптов сторонних программ-отладчиков настоятельно рекомендуется предварительно вводить в скрипт точку останова (Breakpoint).

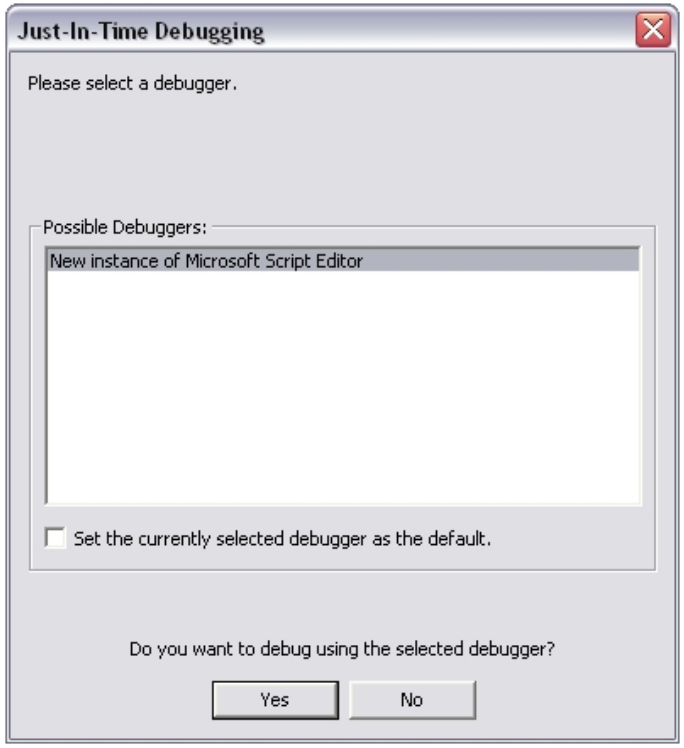
Ввод точки останова выполняется путем добавления в скрипт команды debugger;. При этом выполнение скрипта будет приостановлено в указанном командой debugger; месте и автоматически запустится программа-отладчик.

> **Примечание.**
> В программировании точкой останова (Breakpoint)называют преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика.

Запуск скрипта со сторонней программой-отладчиком может выполнен только по тестовому событию.

Для запуска скрипта с отладкой в сторонней программе-отладчике требуется выполнить следующие действия:

1. Разработать скрипт и добавить в него команду debugger;.
2. Создать тестовое событие для запуска скрипта.
3. Выбрать в меню **Отладка и редактирование** команду **Тестовый пуск с Отладчиком**.
4. На экран будет выведено диалоговое окно **Just In Time Debugging**. В данном окне из списка программ-отладчиков, установленных на компьютере, требуется выбрать необходимую программу.



5. Подтвердить выбор программы-отладчика нажатием кнопки **Yes**.
6. В том случае, если скрипт успешно пройдет проверку синтаксиса и до точки останова (команды debugger;) не будут обнаружены ошибки выполнения скрипта, будет произведен запуск сторонней программы-отладчика. При этом выполнение скрипта будет приостановлено в указанном точкой останова месте.

Пример. Скрипт с использованием точки останова после запуска макрокоманды №1.

```
if (Event.SourceType== "MACRO" && Event.SourceId=="1" && Event.Action == "RUN"); //запуск макрокоманды №1
{
 debugger; //точка останова
 DebugLogString ("Hello world");
}
```

Примеры скриптов на языке JScript

На странице:

- Пример 1. Визуализация работы детектора длины очереди в окне Монитора видеонаблюдения
- Пример 2. Визуализация работы детектора подсчета посетителей в окне Монитора видеонаблюдения

Для иллюстрации возможных областей применения скриптов на языке JScript ниже приведены примеры, которые могут использоваться для создания на основе объекта **Скрипт** дополнительных функций в системе.

Пример 1. Визуализация работы детектора длины очереди в окне Монитора видеонаблюдения

Для корректной работы скрипта в ПК *Интеллект* предварительно должны быть созданы и настроены объекты: **Детектор длины очереди** (входит в состав пакета детекторов Detector Pack), **Камера** и **Титрователь** (ниже вместо символов N, M, L нужно подставить соответствующие номера Детектора длины очереди, Камеры и Титрователя).

```
//Считывание события по длине очереди
if (Event.SourceType == "OCCUPANCY_COUNTER" && Event.SourceId == "N" && Event.Action == "OCCUPANCY")  //N - Номер Детектора длины очереди
{
  var n=Event.GetParam("occupancy");
//Отображение длины очереди через Титрователь в Мониторе
  DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Номер Камеры  L - Номер Титрователя
  DoReactStr("CAM","M","ADD_SUBTITLES","command<Длина очереди: "+n+" чел.\r>,page<BEGIN>,title_id<L>"); //M, L - то же
}
```

В результате при отображении соответствующей камеры в мониторе, на видеоизображение будет накладываться текстовое сообщение о текущей длине очереди.

Настройки шрифта, цвета и положения надписи настраиваются на панели настройки объекта **Титрователь**.

Пример 2. Визуализация работы детектора подсчета посетителей в окне Монитора видеонаблюдения

Для корректной работы скрипта в ПК *Интеллект* предварительно должны быть созданы и настроены объекты: **Детектор подсчета посетителей** (входит в состав пакета детекторов Detector Pack), **Камера**, **Титрователь** и **Макрокоманда** (ниже вместо символов N, M, L, P нужно подставить соответствующие номера Детектора подсчета посетителей, Камеры, Титрователя и Макрокоманды).

```
//Считывание события и подсчет вошедших посетителей
if (Event.SourceType == "PEOPLE_COUNTER" && Event.SourceId == "N" && Event.Action == "IN") //N - Номер Детектора подсчета посетителей
  {
    i = Itv_var("counter_i");
    k = Itv_var("counter_k");
    i++;
    Itv_var("counter_i")=i;
//Отображение количества посетителей через Титрователь в Мониторе

    DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Номер Камеры  L - Номер Титрователя
    DoReactStr("CAM","M","ADD_SUBTITLES","command<Кол-во посетителей (вх./вых.): "+i+" / "+k+"\r>,page<BEGIN>,title_id<L>"); //M, L - то же

  }
//Считывание события и подсчет вышедших посетителей
if (Event.SourceType == "PEOPLE_COUNTER" && Event.SourceId == "N" && Event.Action == "OUT") //N - Номер Детектора подсчета посетителей
  {
    i = Itv_var("counter_i");
    k = Itv_var("counter_k");
    k++;
    Itv_var("counter_k")=k;
//Отображение количества посетителей через Титрователь в Мониторе

   DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Номер Камеры  L - Номер Титрователя
   DoReactStr("CAM","M","ADD_SUBTITLES","command<Кол-во посетителей (вх./вых.): "+i+" / "+k+"\r>,page<BEGIN>,title_id<L>"); //M, L - то же
  }
//Обнуление счетчика по Макрокоманде (предварительно в Интеллекте должна быть создана Макрокоманда)
if (Event.SourceType == "MACRO" && Event.SourceId == "P" && Event.Action == "RUN") //P - Номер Макрокоманды
  {
    Itv_var("counter_i")=0;
    Itv_var("counter_k")=0;
    i=0;
    k=0;
//Отображение количества посетителей через Титрователь в Мониторе

    DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Номер Камеры  L - Номер Титрователя
    DoReactStr("CAM","M","ADD_SUBTITLES","command<Кол-во посетителей (вх./вых.): "+i+" / "+k+"\r>,page<BEGIN>,title_id<L>"); //M, L - то же
  }
```

В результате при отображении соответствующей камеры в Мониторе, на видеоизображение будет накладываться текстовое сообщение о количестве вошедших и вышедших посетителей.

Настройки шрифта, цвета и положения надписи настраиваются на панели настройки объекта **Титрователь** (см. раздел Настройка вывода титров поверх видеоизображения до кумента Руководство администратора).

Для обнуления счетчика посетителей предварительно на вкладе **Программирование** создается объект **Макрокоманда**, название которой можно для удобства изменить, например, на «Обнуление счетчика посетителей».

Макрокоманду обнуления можно запускать как вручную через главное меню ПК *Интеллект*, так и автоматически в любое заданное время (для этого используется таблица **Соб ытия** на панели настройки объекта **Макрокоманда**, где необходимо указать предварительно настроенный объект **Временная зона**). Подробные сведения об использовании объектов **Макрокоманда** и **Временная зона** изложены в документе Руководство Администратора.

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.

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 утилиты Редактор-Отладчик

Назначение утилиты Редактор-Отладчик

Утилита *Редактор-Отладчик* предназначена для создания, редактирования и отладки скриптов в программном комплексе *Интеллект*.

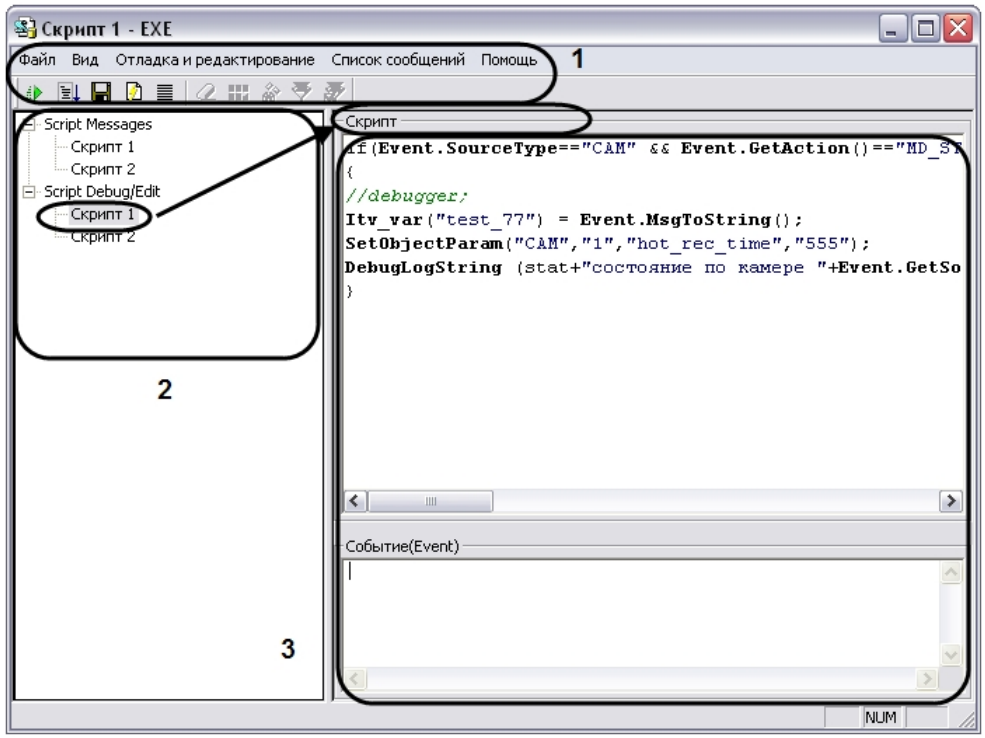
Утилита *Редактор-Отладчик* обеспечивает выполнение следующих задач:

1. создание и редактирование скриптов с использованием встроенного текстового редактора;
2. отладка скрипта посредством встроенного отладочного окна;
3. использование фильтра вывода информации в отладочном окне;
4. создание и использование тестового события для отладки скрипта;
5. сохранение скрипта на жесткий диск;
6. загрузка скрипта с жесткого диска.

Описание интерфейса утилиты Редактор-Отладчик

Интерфейс утилиты Редактор-Отладчик

Пользовательский интерфейс утилиты *Редактор-отладчик* представлен главным меню и панелью инструментов (1), списком объектов (2) и панелью редактирования/просмотра (3).



Вкладка Script Debug/Edit

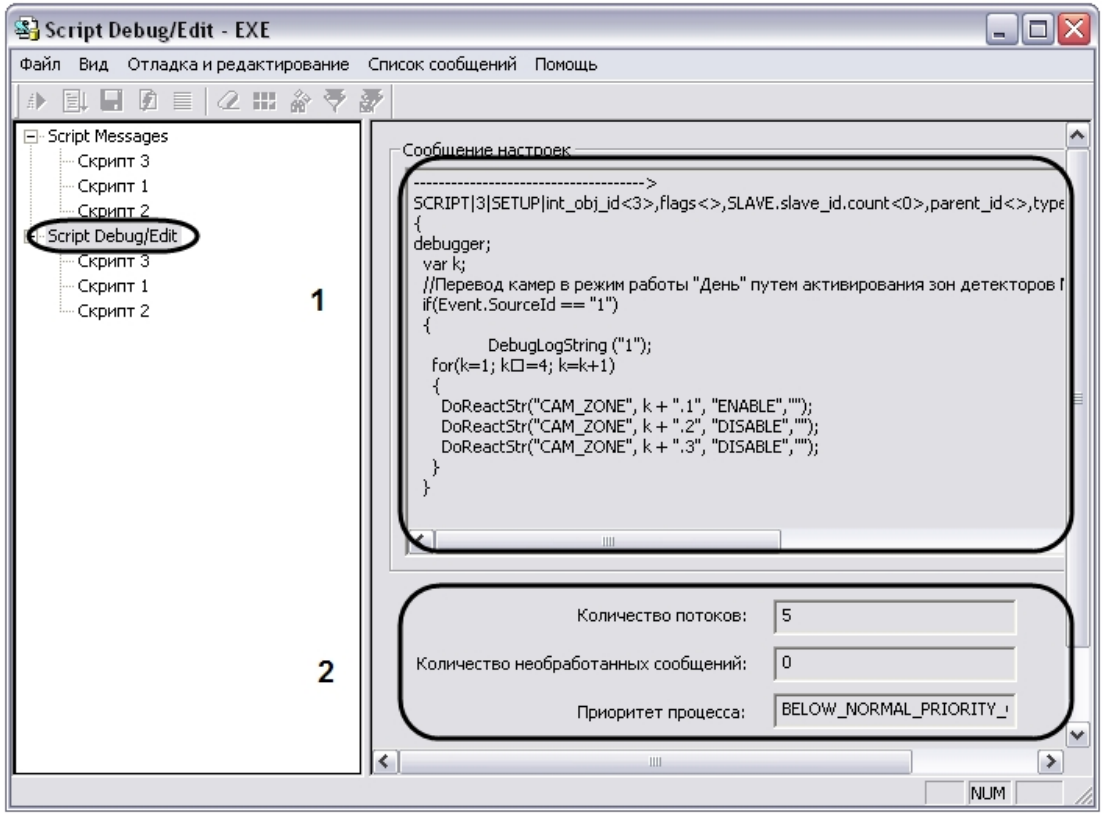
На странице:

- Описание интерфейса вкладки Script Debug/Edit
- Описание интерфейса объекта Скрипт (вкладка Script Debug/Edit)

Описание интерфейса вкладки Script Debug/Edit

Вкладка **Script Debug/Edit** предназначена для редактирования скриптов и создания тестовых событий.

Интерфейс вкладки **Script Debug/Edit** представлен на рисунке.



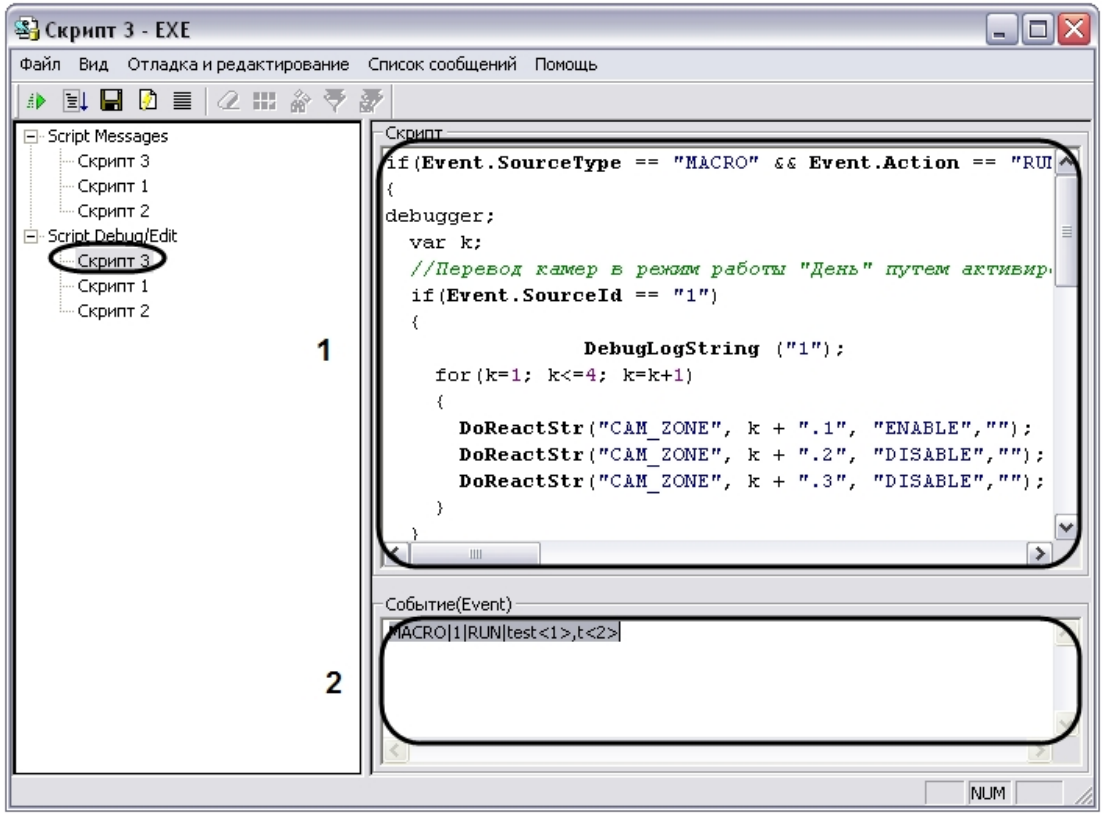
Описание интерфейса вкладки **Script Debug/Edit** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Сообщение настроек	Автоматически	Информация об инициализации объектов **Скрипт** в системе.	Латинский алфавит, кириллица и служебные символы	-	-
2	Дополни-тельная информация	Автоматически	Дополнительная информация о скриптах.	Латинский алфавит, кириллица и служебные символы	-	-

Описание интерфейса объекта Скрипт (вкладка Script Debug/Edit)

Объект **Скрипт** вкладки **Script Debug/Edit** предназначен для создания и редактирования скриптов и тестового событий.

Интерфейс объекта **Скрипт** представлен на рисунке.



Описание интерфейса объекта **Скрипт** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Скрипт	Ввод значения в поле	Содержит текст скрипта.	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов
2	Событие **Event**	Ввод значения в поле	Содержит текст тестового события.	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов

Вкладка Script Messages

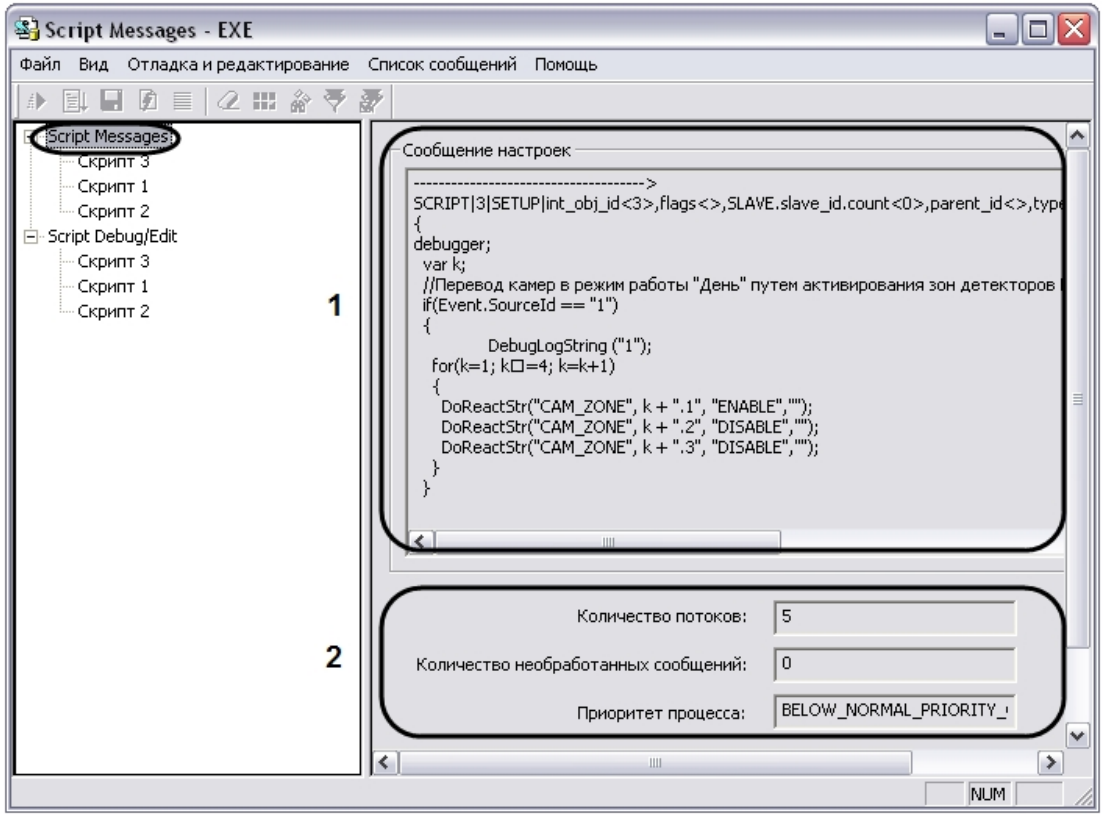
На странице:

- Описание интерфейса вкладки Script Messages
- Описание интерфейса объекта Скрипт (вкладка Script Messages)

Описание интерфейса вкладки Script Messages

Вкладка **Script Messages** предназначена для отображения отладочных окон скриптов.

Интерфейс вкладки **Script Messages** представлен на рисунке.



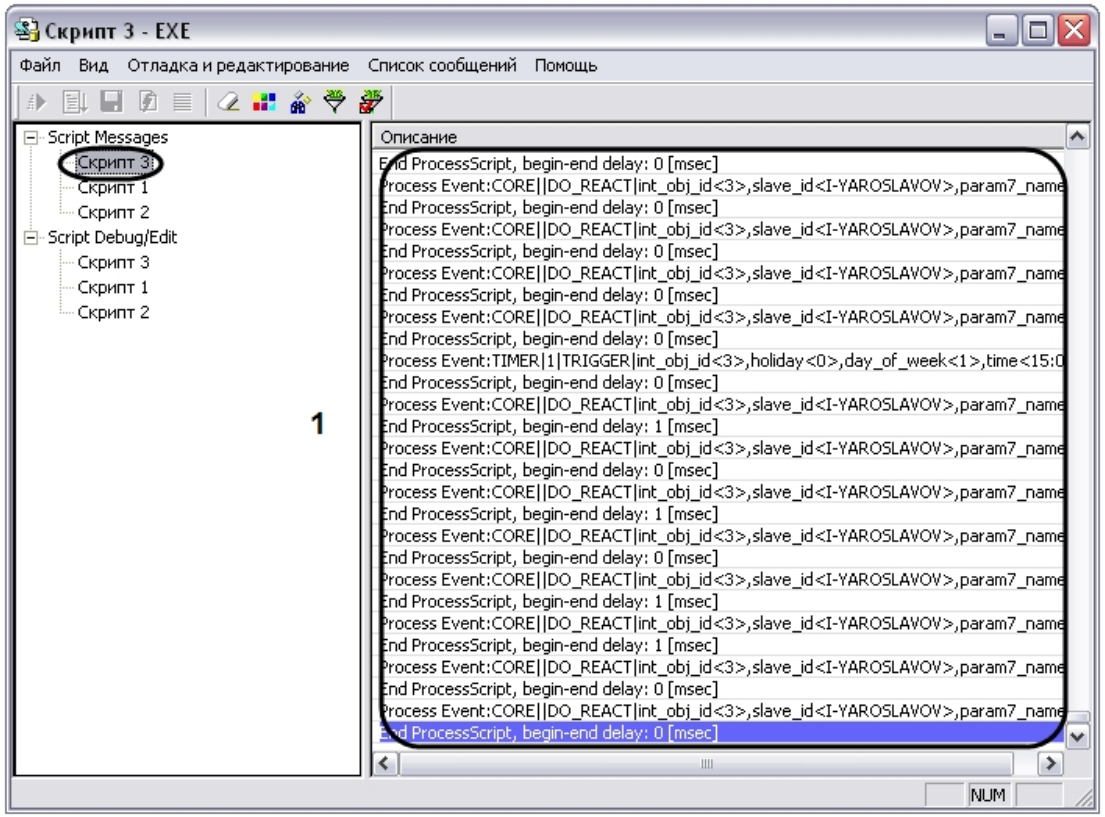
Описание интерфейса вкладки **Script Messages** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Сообщение настроек	Автоматически	Информация об инициализации объектов **Скрипт** в системе.	Латинский алфавит, кириллица и служебные символы	-	-
2	Дополни-тельная информация	Автоматически	Дополнительная информация о скриптах.	Латинский алфавит, кириллица и служебные символы	-	-

Описание интерфейса объекта Скрипт (вкладка Script Messages)

Объект **Скрипт** вкладки **Script Messages** предназначен для отображения системных, тестовых и пользовательских событий, относящихся к созданным в ПК *Интеллект* скриптам.

Интерфейс объекта **Скрипт** представлен на рисунке.



Описание интерфейса объекта **Скрипт** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Описание	Автоматически	Отображает информацию о событиях, происходящих в системе.	Латинский алфавит, кириллица и служебные символы	На задано	По умолчанию в данном списке отображается 200 строк. Изменить это значение можно при помощи ключа реестра DebugMaxLines (см. Справочник ключей реестра).

Главное меню

Описание интерфейса главного меню

Главное меню утилиты *Редактор-Отладчик* предназначено для вызова команд, реализуемых утилитой. Команды разделены по функциональным признакам на группы и размещены в следующих пунктах меню: **Файл, Вид**, **Отладка и редактирование**, **Список сообщений**, **Помощь.**

Описание пунктов главного меню утилиты представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Файл	Выбор команды из раскрыва-ющегося списка	Содержит список команд, обеспечивающих сохранение и загрузку скриптов, завершение работы с утилитой.	-	-	
2	Вид	Выбор команды из раскрыва-ющегося списка	Содержит список команд, обеспечивающих отображение в окне утилиты панели инструментов и строки состояния.	-	-	-
3	Отладка и редактиро-вание	Выбор команды из раскрыва-ющегося списка	Содержит список команд, обеспечивающих процесс отладки скрипта.	-	-	-
4	Список сообщений	Выбор команды из раскрыва-ющегося списка	Содержит список команд, позволяющих изменять параметры отображения сообщений в отладочных окнах утилиты.	-	-	-
5	Помощь	Выбор команды из раскрыва-ющегося списка	Содержит команду **О программе...**. По данной команде осуществляется вызов окна, содержащего общие сведения об утилите *Редактор-Отладчик*.	-	-	-

Описание пункта главного меню Файл

Пункт главного меню **Файл** предназначен для вызова команд сохранения и загрузки скриптов, завершения работы с утилитой.

Описание элементов пункта главного меню **Файл** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Сохранить в базе	Выбор команды	Осуществляет сохранение скрипта в соответствующий ему системный объект.	-	-	-
2	Сохранить на диск	Выбор команды	Осуществляет сохранение скрипта в текстовый файл на жестком диске.			
3	Загрузить с диска	Выбор команды	Осуществляет загрузку скрипта из выбранного текстового файла в редактор утилиты.	-	-	-

4	Выход	Выбор команды	Команда предназначена для завершения работы с утилитой и закрытия ее диалогового окна.	-	-	-

Описание пункта главного меню Вид

Пункт главного меню **Вид** предназначен для вызова команд включающих и отключающих отображение в окне утилиты панели инструментов и строки состояния.

Описание элементов пункта главного меню **Вид** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значе-ние по умол-чанию	Диапазон значений
1	Инструменты	Установка флажка	Включает или отключает отображение панели инструментов в окне утилиты.	Тип Boolean	Да	Да – панель инструментов отображается. Нет – панель инструментов не отображается.
2	Строка состояния	Установка флажка	Включает или отключает отображение строки состояния в нижней части окна утилиты.	Тип Boolean	Да	Да – строка состояния отображается. Нет – строка состояния не отображается.

Описание пункта главного меню Отладка и редактирование

Пункт главного меню **Отладка и редактирование** предназначен вызова команд отладки скриптов.

Описание элементов пункта главного меню **Отладка и редактирование** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Тестовый пуск	Выбор команды	Производит запуск скрипта по тестовому событию.	-	-	
2	Тестовый запуск с отладчиком	Выбор команды	Производит запуск скрипта по тестовому событию с использованием сторонней программы-отладчика.	-	-	-
3	Редактиро-вать тестовое событие	Выбор команды	Вызывает диалоговое окно, предназначенное для редактирования тестового события.	-	-	-
4	Сводная информация	Выбор команды	Вызывает отладочное окно **Информация от потока**, в котором отображаются системные, тестовые и пользовательские сообщения, относящиеся только к отлаживаемому скрипту.	-	-	-
5	Перейти к строке	Выбор команды	Вызывает диалоговое окно, предназначенное для задания номеров символа и строки в тексте скрипта, к которым требуется осуществить переход.	-	-	-

Описание элементов пункта главного меню Список сообщений

Пункт главного меню **Список сообщений** предназначен для изменения параметров отображения сообщений в отладочном окне.

Описание элементов пункта меню **Список сообщений** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Очистить	Выбор команды	Очищает поле **Описание** отладочного окна.	-	-	
2	Цвета	Выбор команды	Вызывает окно, предназначенное для задания слов (или других последовательностей символов), строки, содержащие которые, требуется выделять цветом в поле **Описание** отладочного окна.	-	-	-
3	Поиск	Выбор команды	Осуществляет поиск слова (или других последовательностей символов) в поле **Описание** отладочного окна.	-	-	-
4	Установить фильтр	Выбор команды	Вызывает окно, с помощью которого осуществляется включение и настройка фильтра. Строки, которые содержат (не содержат) указанные слова должны обязательно быть включены в отладочное окно (или исключены из него).	-	-	-
5	Применить фильтр	Выбор команды	Активирует созданный фильтр.	-	-	-

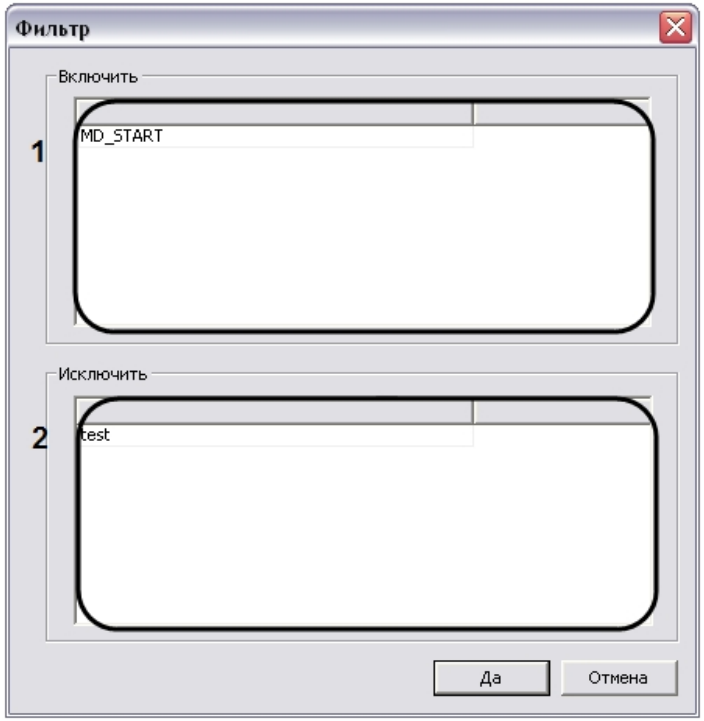
Описание диалогового окна Фильтр

Диалоговое окно **Фильтр** предназначено для включения и настройки фильтров сообщений, отображаемых в поле **Описание отладочного окна**.

Интерфейсное окно **Фильтр** вызывается двумя способами:

1. нажатием кнопки **Редактировать тестовое сообщение**  , расположенной на панели инструментов утилиты *Редактор-Отладчик*;
2. по команде **Отладка и редактирование** -> **Редактировать тестовое сообщение**.

Интерфейс окна **Фильтр** представлен на рисунке.



Описание интерфейса диалогового окна **Фильтр** представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Включить	Ввод значения в поле	Строки, содержащие указанные в данном поле слова (или другие последовательности символов), будут отображаться в отладочном окне.	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов
2	Исключить	Ввод значения в поле	Строки, содержащие указанные в данном поле слова (или другие последовательности символов), будут исключены из отладочного окна.	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов

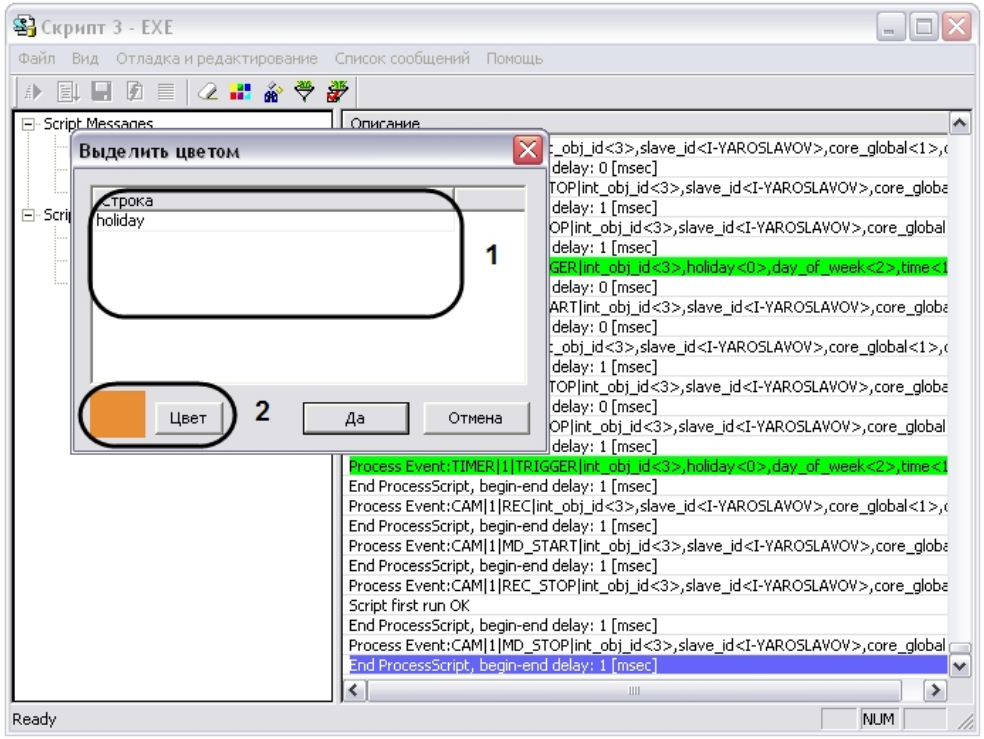
Описание диалогового окна Выделить цветом

Диалоговое окно **Выделить цветом** предназначено для настройки выделения в отладочном окне цветом строк, содержащих заданные слова.

Диалоговое окно **Выделить цветом** вызывается двумя способами:

1. нажатием кнопки **Цвета**  , расположенной на панели инструментов утилиты *Редактор-Отладчик*;
2. по команде **Список сообщений** -> **Цвета.**

Интерфейс окна **Выделить цветом** представлен на рисунке.



Описание элементов диалогового окна **Фильтр** представлено в таблице.

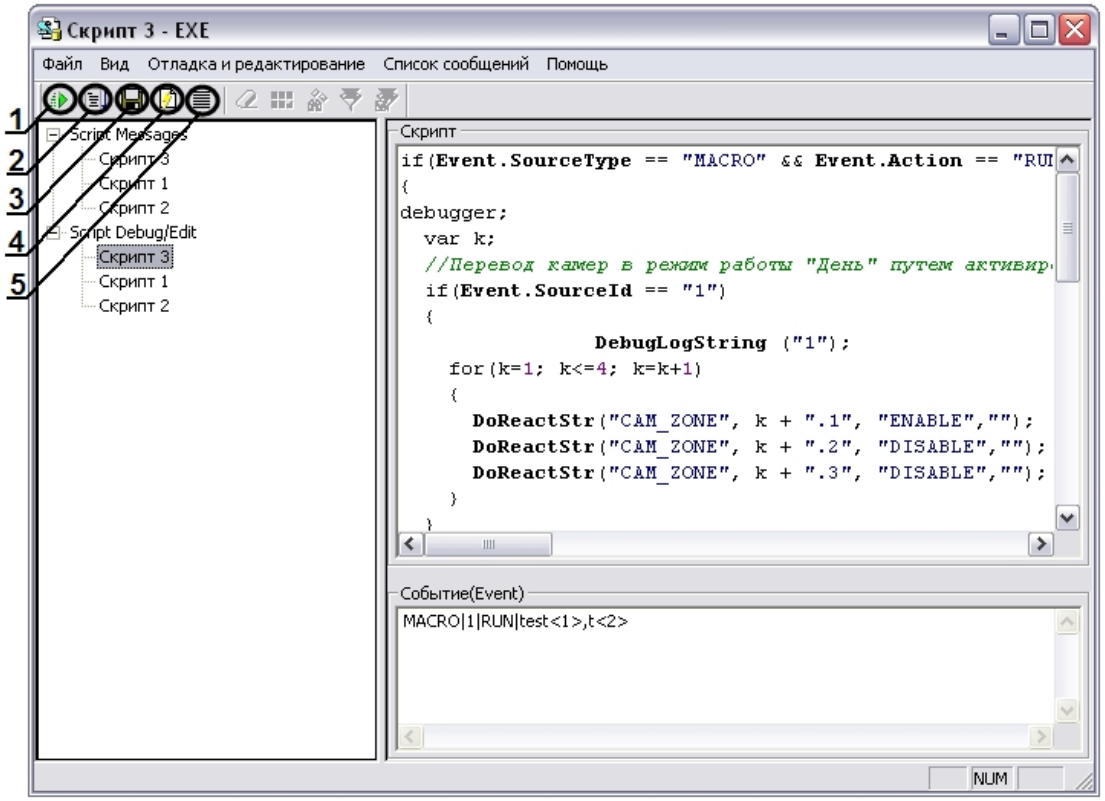
№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Строка	Ввод значения в поле	Задает слова (или другие последовательности символов) для выделения содержащих их строк в отладочном окне.	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограничен-ное количество символов
2	Цвет	Выбор из списка значений	Задает цвет выделения строк в отладочном окне.	RGB	Серый	Цветовой диапазон RGB формата

Описание панели инструментов утилиты Редактор-Отладчик

Панель инструментов утилиты *Редактор-Отладчик* предназначена для вызова часто используемых функций утилиты.

Панель инструментов утилиты *Редактор-Отладчик* функционирует в двух режимах: с активными кнопками управления скриптами или с активными кнопками управления функциями отладочного окна. Активность кнопок зависит от того, какая вкладка утилиты *Редактор-Отладчик* является активной в данный момент: вкладка **Script Debug\Edit**, используемая для редактирования скриптов, либо вкладка **Script Messages**, используемая для просмотра сообщений в отладочном окне.

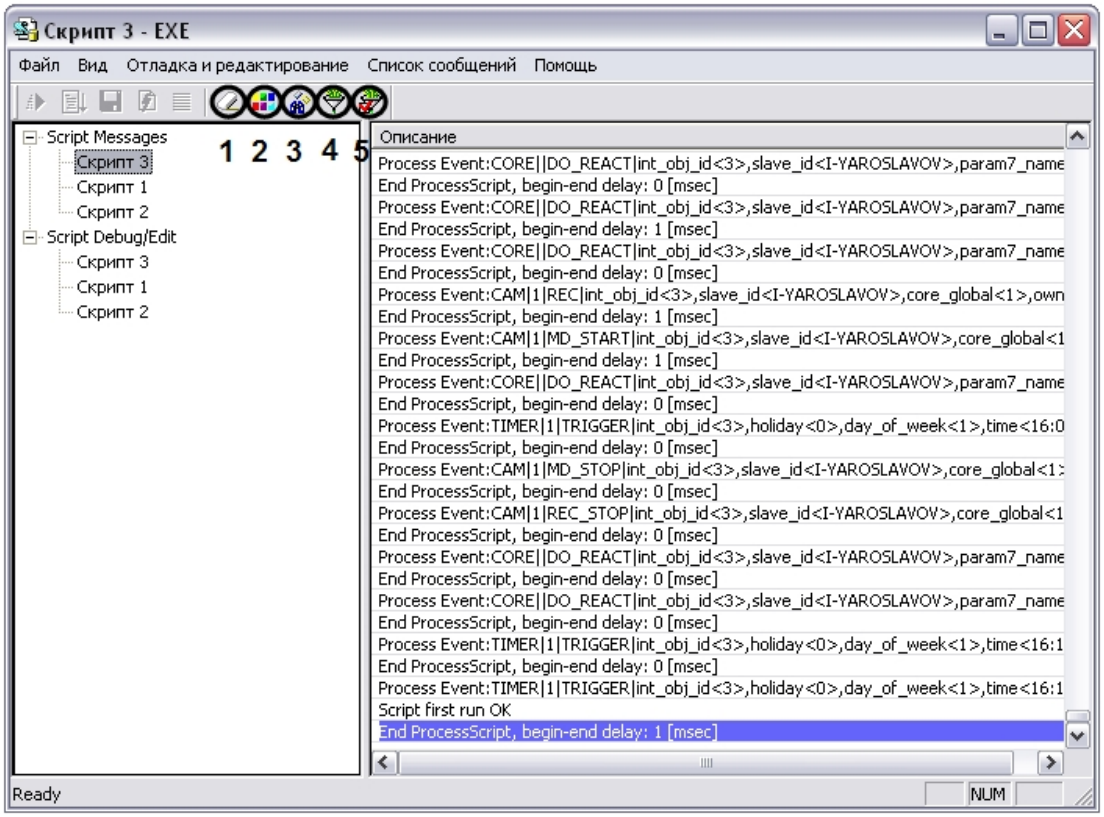
Интерфейс панели инструментов *Редактор-Отладчик* в режиме редактирования скрипта представлен на рисунке.



Описание интерфейсов панели инструментов утилиты *Редактор-Отладчик* в режиме редактирования скриптов представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Тестовый запуск	Нажатием кнопки	Производит запуск скрипта по тестовому событию.	-	-	-
2	Тестовый запуск с отладчиком	Нажатием кнопки	Производит запуск скрипта по тестовому событию с использованием сторонней программы-отладчика.	-	-	-
3	Сохранить	Нажатием кнопки	Осуществляет сохранение скрипта в системный объект **Скрипт**.	-	-	-
4	Редактиро-вать тестовое событие	Нажатием кнопки	Вызывает диалоговое окно, предназначенное для редактирования тестового события.	-	-	-
5	Сводная информация	Нажатием кнопки	Вызывает отладочное окно **Информация от потока**, в котором отображаются системные, тестовые и пользовательские сообщения, относящиеся только к отлаживаемому скрипту.	-	-	-

Интерфейс панели инструментов утилиты *Редактор-Отладчик* в режиме работы с отладочным окном представлен на рисунке.



Описание интерфейса панели инструментов утилиты *Редактор-Отладчик* в режиме работы с отладочным окном представлено в таблице.

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Очистить	Нажатием кнопки	Очищает поле **Описание** отладочного окна.	-	-	
2	Цвета	Нажатием кнопки	Вызывает окно, предназначенное для задания слов (или других последовательностей символов), строки, содержащие которые, требуется выделять цветом в поле **Описание** отладочного окна.	-	-	-
3	Поиск	Нажатием кнопки	Осуществляет поиск слова (или других последовательностей символов) в поле **Описание** отладочного окна.	-	-	-
4	Установить фильтр	Нажатием кнопки	Вызывает окно, с помощью которого осуществляется включение и настройка фильтра. Строки, которые содержат (не содержат) указанные слова должны обязательно быть включены в отладочное окно (или исключены из него).	-	-	-
5	Применить фильтр	Нажатием кнопки	Активирует созданный фильтр.	-	-	-

Приложение 2. Создание виртуальных объектов с возможностью задавать события, реакции и состояния

Назначение виртуальных объектов и их реализация в ПК Интеллект

Виртуальные объекты представляют собой программную эмуляцию новых объектов ПК *Интеллект* и позволяют настраивать свои состояния, реакции, события. Работа с виртуальными объектами осуществляется при помощи скриптов, макрокоманд и макрособытий.

Создание виртуальных объектов выполняется с использованием утилит ddi.exe и CustomTypeEditor.exe, расположенных в каталоге <Директория установки ПК *Интеллект*>\Tools.

В разделе Пример создания виртуального объекта рассмотрен пример создания двух типов виртуальных объектов, которые можно использовать, например, для отображения состояния детектора оставленных предметов на карте, или для отображения любых других пользовательских состояний. При этом изменение состояний объекта осуществляется при помощи макрокоманд, скриптов или через IIDK.

Пример создания виртуального объекта

В данном разделе описана последовательность действий для создания следующих виртуальных объектов:

1. Тип CUSTOM, с родительским типом SLAVE (Компьютер).
2. Тип CUSTOM_CHILD, с родительским типом CUSTOM (см. п.1).

Объект типа CUSTOM имеет набор свойств:

1. Параметры custom_param1, custom_param2
2. События: ALARM, INFO, ON, OFF
3. Реакции: ON, OFF
4. Состояния: ON, OFF
5. Машина состояний:
    a. На событие ON выставить состояние ON
    b. На событие OFF выставить состояние OFF

Тип объекта CUSTOM_CHILD создается для демонстрации иерархической структуры и не имеет пользовательских параметров, событий, реакций и состояний.

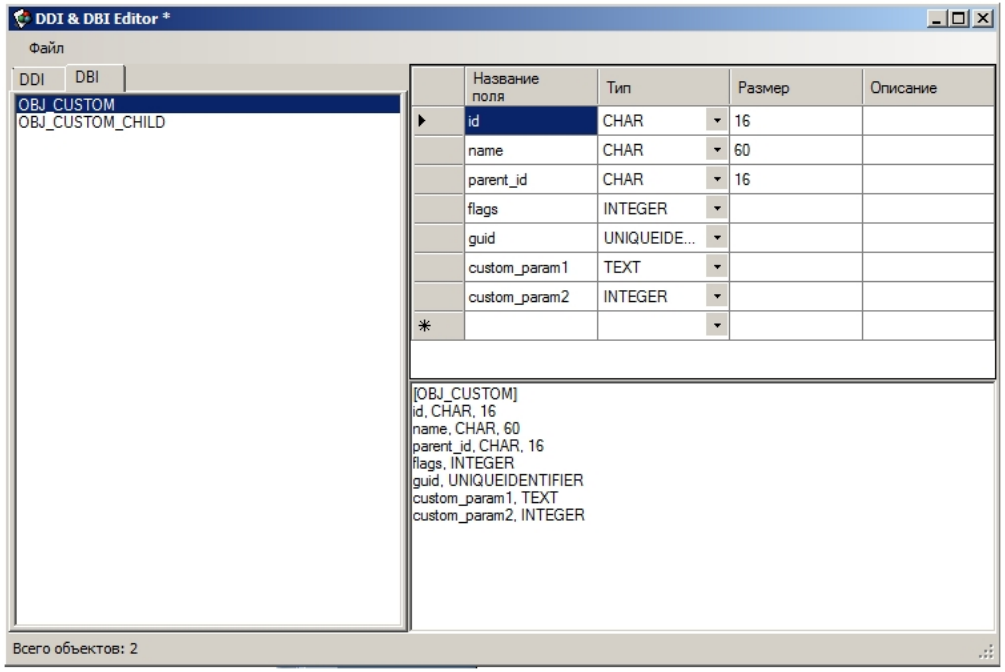
Подготовка файла dbi

Подготовка файла dbi осуществляется при помощи утилиты ddi.exe. Работа с ней подробно описана в документе Руководство Администратора, в разделе Утилита редактирования шаблонов баз данных и файла внешних настроек ddi.exe.

Создание файла dbi для типов объектов CUSTOM и CUSTOM_CHILD выполняется в следующей последовательности:

1. Запустить утилиту ddi.exe (см. Руководство Администратора, раздел Утилита редактирования шаблонов баз данных и файла внешних настроек ddi.exe).
2. Перейти на вкладку **DBI**.
3. Создать два объекта, OBJ_CUSTOM и OBJ_CUSTOM_CHILD, как показано на рисунке ниже.

> ⚠ **Внимание!**
> Названия объектов (таблиц) должны иметь вид OBJ_<тип объекта>.



> **Примечание.**
> Параметры id, name, parent_id, flags, guid - являются обязательными для всех объектов. custom_param1, custom_param2 – пользовательские параметры.

4. Сохранить изменения при помощи команды **Сохранить** в меню **Файл**. Сохраненный файл должен иметь расширение dbi и располагаться в директории установки ПК *Интеллект*, например C:\Program Files (x86)\Интеллект\intellect.custom.dbi

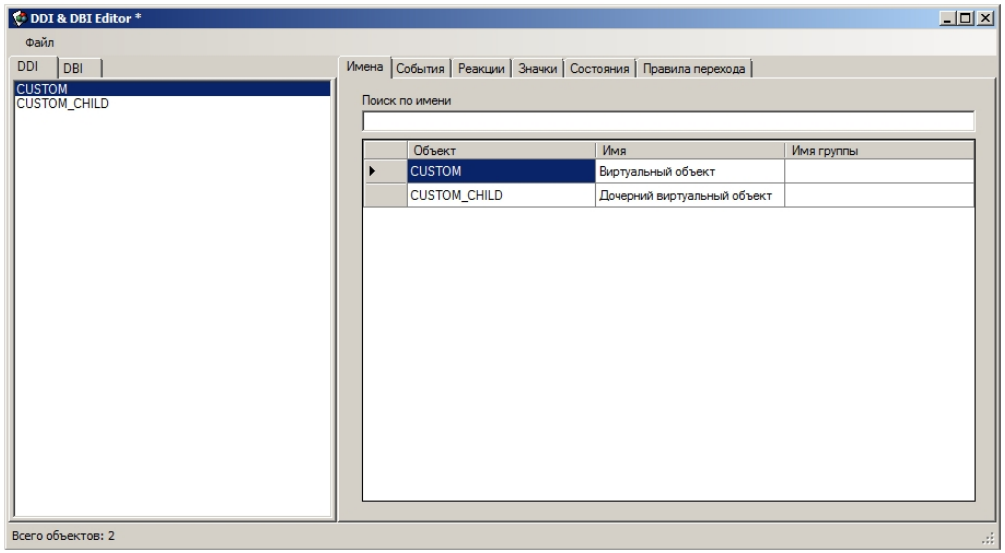
Подготовка файла dbi завершена.

Подготовка файла ddi

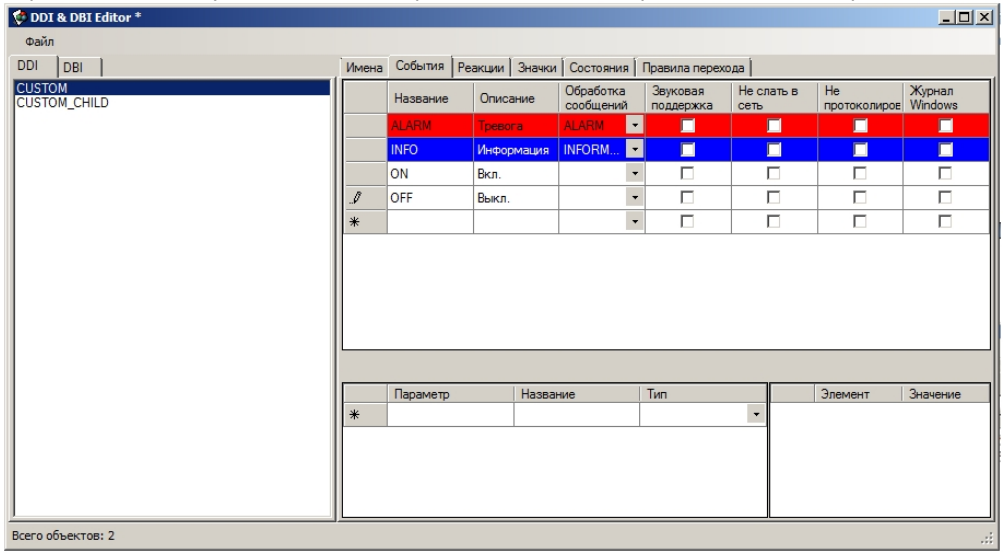
Подготовка файла ddi осуществляется при помощи утилиты ddi.exe. Работа с ней подробно описана в документе Руководство Администратора, в разделе Утилита редактирования шаблонов баз данных и файла внешних настроек ddi.exe.

Создание файла ddi для типов объектов CUSTOM и CUSTOM_CHILD выполняется в следующей последовательности:

1. Запустить утилиту ddi.exe (см. Утилита редактирования шаблонов баз данных и файла внешних настроек ddi.exe).
2. На вкладке **DDI** создать два объекта, CUSTOM и CUSTOM_CHILD, как показано на рисунке ниже.

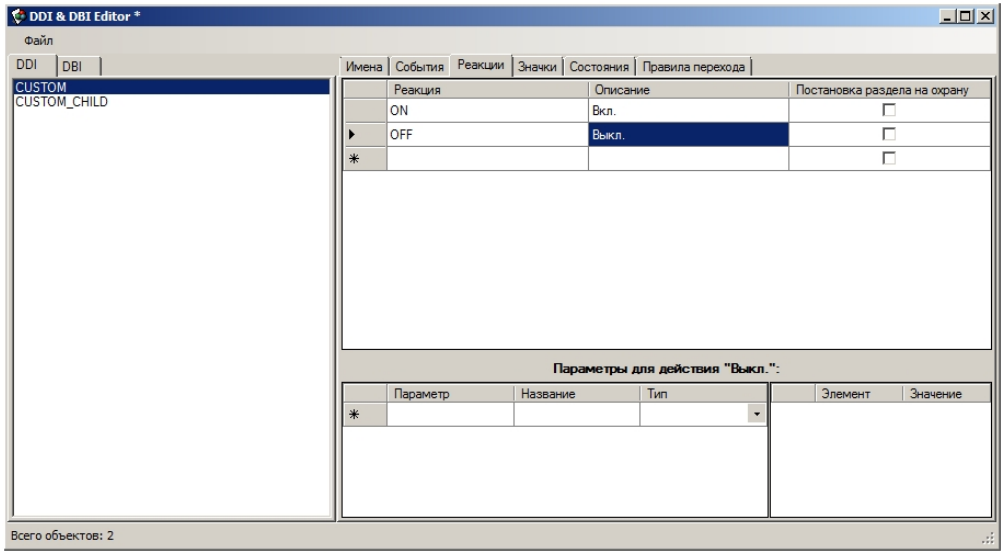


3. Перейти на вкладку **События** и настроить события, которые должны поддерживаться объектом (см. рисунок).

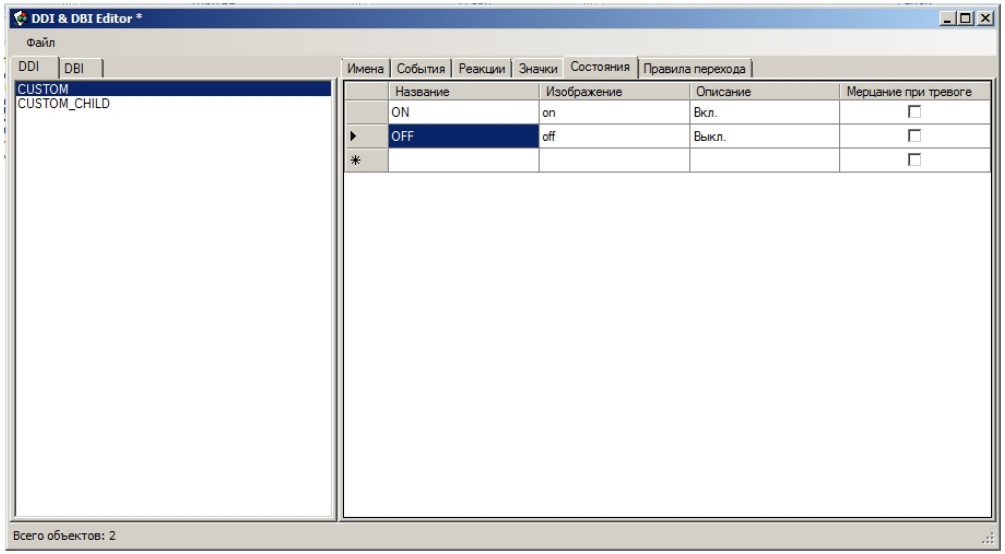


4. Перейти на вкладку **Реакции** и настроить реакции, которые должен поддерживать объект (см. рисунок).

> **Примечание.**
> Реакции виртуальных объектов автоматически конвертируются в события. Иными словами, при поступлении реакции виртуальный объект автоматически сгенерирует соответствующее событие.

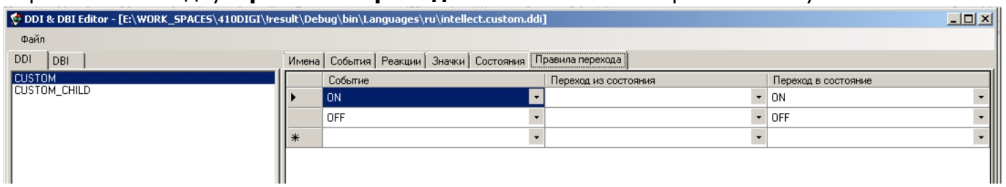


5. Перейти на вкладку **Состояния** и описать состояния, которые может принимать объект. В данном примере рассмотрено два состояния – ON и OFF.



Примечание.
В столбце **Изображение** указывается постфикс имени файла изображения, расположенного в папке <Директория установки ПК *Интеллект*>\Bmp. Например, для объекта CUSTOM это будут файлы custom_off.bmp и custom_on.bmp, соответствующие состояниям ON и OFF. Эти файлы будут использованы модулем карта.

6. Перейти на вкладку **Правила перехода состояний** и настроить логику изменения состояния объекта.



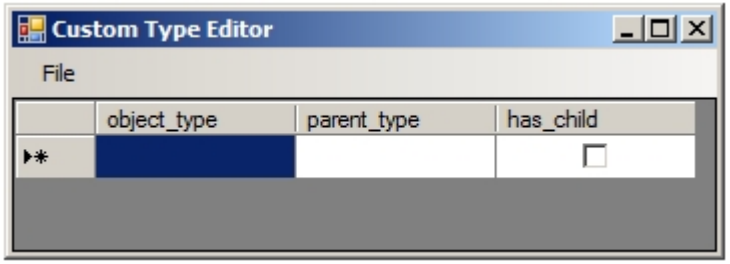
Правила перехода состояний — это простая машина состояний, входным действием является событие, а результатом — состояние.

В рассматриваемом примере используется безусловный переход: если пришло событие CUSTOM||ON, производится переход в состояние ON, если пришло событие CUSTOM||OFF — в состояние OFF.

Подготовка файла ddi завершена.

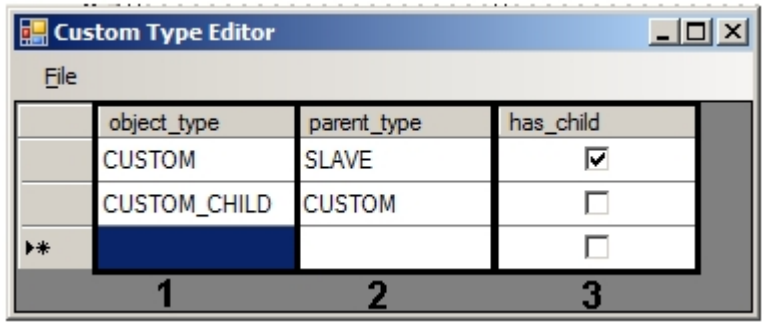
Подготовка файла xml

Подготовка файла xml осуществляется при помощи утилиты CustomTypeEditor.exe, расположенной в папке <Директория установки ПК *Интеллект*>\Tools. Общий вид окна утилиты представлен на рисунке.



Создание файла xml для виртуального объекта выполняется следующим образом:

1. В поле **object_type** ввести название типа объекта (1).



2. В поле **parent_type** ввести название родительского типа (2).
3. В случае, если у типа объекта будут иметься дочерние типы, установить флажок в столбце **has_child** (3).
4. Повторить шаги 1-3 для всех типов объектов.
5. Сохранить файл в директории установки ПК *Интеллект* при помощи команды меню **File** – **Save**.

Создание файла xml завершено. Созданный файл будет иметь следующее содержание:

```
<?xml version="1.0" standalone="yes"?>
<objects>
   <object>
      <object_type>CUSTOM</object_type>
      <parent_type>SLAVE</parent_type>
      <has_child>1</has_child>
</object>
   <object>
      <object_type>CUSTOM_CHILD</object_type>
<parent_type>CUSTOM</parent_type>
   </object>
</objects>
```

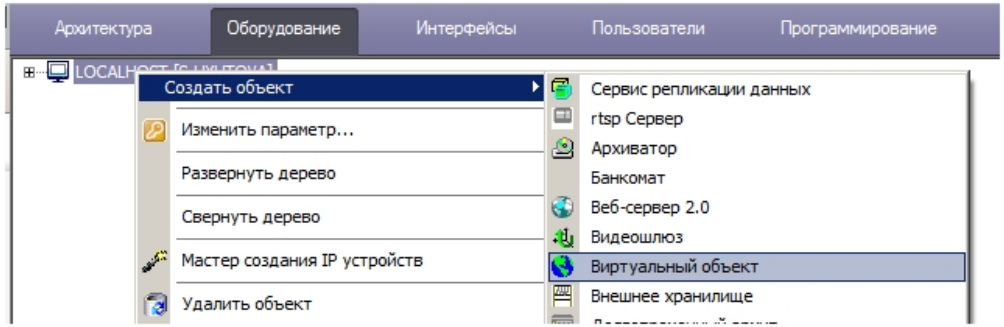
При необходимости можно редактировать его и вручную.

Создание и использование виртуального объекта в ПК Интеллект

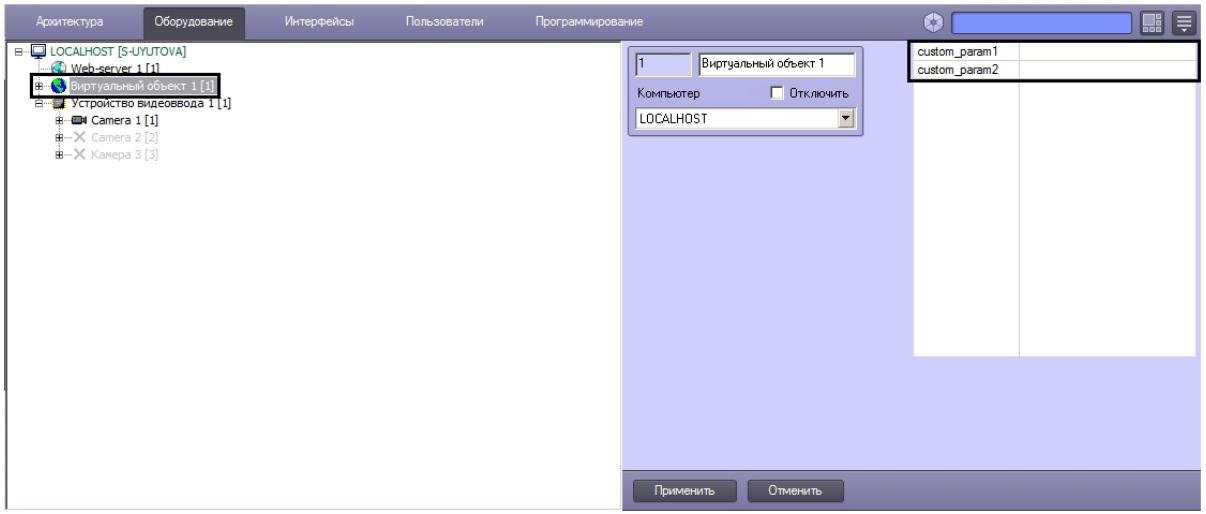
На странице:

- Отображение на карте
- Использование в макрокомандах
- Пример программы на языке JScript для изменения состояния виртуального объекта

После подготовки файлов dbi, ddi и xml появится возможность создать в дереве оборудования ПК *Интеллект* объекты нового типа наряду со стандартными объектами.

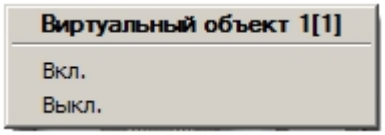


На панели настройки созданного виртуального объекта будут отображены пользовательские параметры – в рассматриваемом примере это custom_param1 и custom_param2. Их значения можно задавать в таблице.



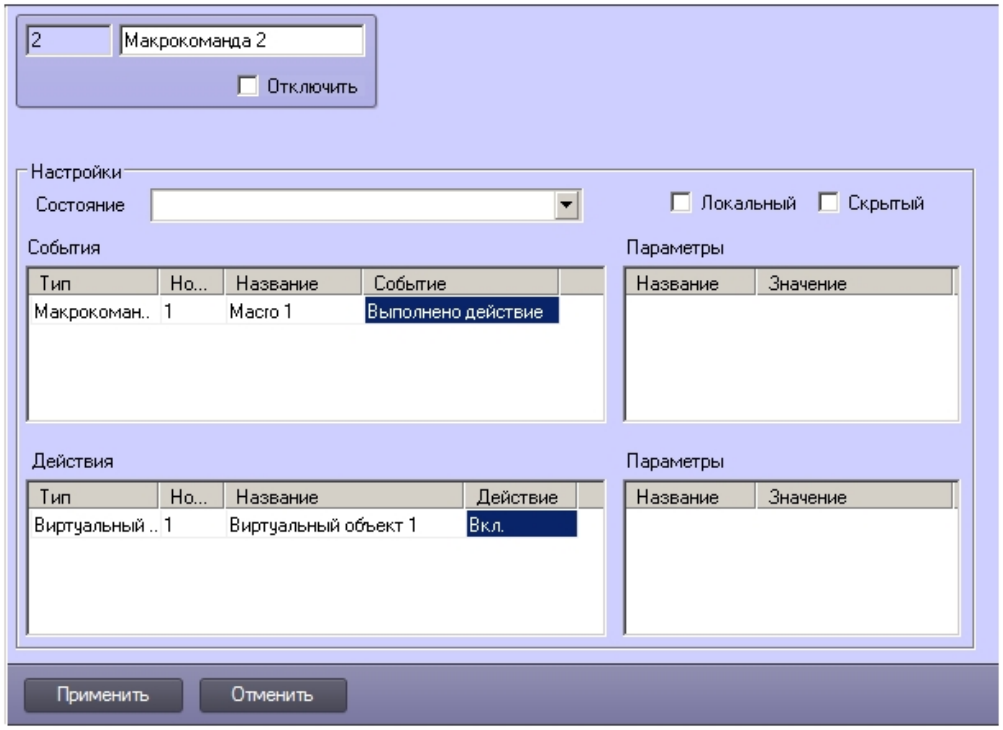
Отображение на карте

После создания в дереве оборудования объект можно размещать на карте и выполнять заданные реакции из контекстного меню объекта (см. Настройка интерактивной карты для индикации состояний и управления системными объектами).



Использование в макрокомандах

После создания виртуального объекта в дереве оборудования имеется также возможность использовать его в макрокомандах.



Примечание.
Реакции виртуальных объектов автоматически конвертируются в события. Таким образом, в рассматриваемом примере, благодаря настроенным правилам перехода состояний (см. Подготовка файла ddi), при выполнении реакции **Вкл.** объект перейдет в соответствующее состояние, и на карте будет отображена иконка, соответствующая данному состоянию.

Пример программы на языке JScript для изменения состояния виртуального объекта

Задача. По макрокоманде 1 изменить состояние виртуального объекта с идентификатором 1 на ON и отобразить на карте соответствующую данному состоянию иконку.

Решение. Поскольку в рассматриваемом примере виртуального объекта настроены правила перехода состояний, при отправке события ON от виртуального объекта его состояние будет автоматические изменено на ON, а на карте будет отображена иконка, указанная в файле ddi для данного состояния (см. Подготовка файла ddi). Скрипт для отправки события ON имеет вид:

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")

{
   var msgevent = CreateMsg();

   msgevent.SourceType = "CUSTOM";

   msgevent.SourceId = "1";

   msgevent.Action = "ON";

   NotifyEvent(msgevent);

}
```