MACROSCOP

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ IP-КАМЕР

ИСПОЛЬЗОВАНИЕ MACROSCOP SDK ДЛЯ РАЗРАБОТКИ ВНЕШНИХ МОДУЛЕЙ

Координаты службы технической поддержки:

Телефон: (+7 342) 215-09-78
E-mail: support@macroscop.com
ICQ: 604028140
Skype: macroscop.support

Оглавление

1 Общие сведения о MACROSCOP SDK

MACROSCOP SDK – это инструментарий, позволяющий создавать программное обеспечение, именуемое плагинами (внешними модулями), позволяющее расширять существующие функциональные возможности программного комплекса **MACROSCOP**.

Данный инструментарий предназначен для .NET программистов, желающих создавать плагины для **MACROSCOP**. Все исходные файлы инструментария и примеров написаны для .NET на языке C#. В качестве среды разработки предполагается использование Microsoft Visual Studio. Для понимания данного документа требуется владение терминологией **MACROSCOP** на уровне опытного пользователя. При необходимости вы можете обратиться к инструкциям оператора и администратора, поставляемых в комплекте с **MACROSCOP**.

В рамках **MACROSCOP SDK,** каждый плагин представляет собой наследника одного из доступных в инструментарии базовых классов (интерфейсов) и решает определенный ряд задач. На данный момент в инструментарии имеются следующие основные базовые классы (интерфейсы), которые могут быть использованы внешними разработчиками:

- **ExternalAction** *(Плагин-действие).* Базовый класс, позволяющий добавлять новые действия для сценариев и планировщика задач по расписанию.
- **VideoAnalyst** *(Плагин-видеоаналитик).* Базовый класс для осуществления видео аналитики на сервере.
- **MotionDetector** *(Плагин-детектор движения)*. Базовый класс для реализации детектора движения.
- **Tracker** *(Плагин-трекер)*. Базовый класс для создания трекера.
- **RTVisualiser** *(Плагин-визуализатор)*. Базовый класс визуализатора для графического отображения специфической информации на канале в клиенте **MACROSCOP**.
- **ClientMenuItem** *(Плагин-элемент меню).* Базовый класс элемента меню, позволяющий создавать подпункт в меню кнопки «Настройка» в клиенте.
- **EventProcessor** *(Плагин-процессор событий)*. Базовый класс процессора событий. Позволяет регистрировать и генерировать свои события, получать события от **MACROSCOP**, а также выполнять команды в канале. Плагины данного типа могут быть использованы для осуществления интеграции с другими системами.
- **IRealTimeFrameReceiver** *(Плагин-получатель кадров)*. Интерфейс получателя кадров с IP устройств. Позволяет получать видео, звук, данные детекции движения, управлять поворотными камерами.

Все указанные типы базовых классов (интерфейсов), а также некоторые другие вспомогательные сущности подробно рассматривается в соответствующих главах данного документа. Все плагины существуют и работают в рамках канала **MACROSCOP**. Таким образом, все экземпляры плагинов по умолчанию изолированы друг от друга, однако имеется возможность обмениваться данными при необходимости через статические поля плагина. Как правило, все плагины решающие в совокупности одну сложную задачу, находятся в одной сборке .NET. Такая сборка является динамически подключаемой библиотекой (DLL), функционирующей в среде **MACROSCOP**. Подключение сборок и регистрация плагинов происходит на этапе запуска отдельных компонентов (сервер/клиент/конфигуратор) программного комплекса. Более подробно о регистрации плагинов в системе вы можете прочитать в главе 3.

2 Быстрый старт – типовые задачи

1. **Интеграция IP камер**
   Для подключения ip-устройства достаточно реализовать плагин получатель-кадров. Необходимые сведения о данном типе плагина вы найдете в главе 4.8. Каркас плагина находится в папке с примерами, проект Camera.csproj.
2. **Интеграция со СКУД/ОПС, POS-терминалы**
   Интеграция может быть выполнена через плагин-процессор событий, умеющий получать события от **MACROSCOP**, генерировать свои события в процессе взаимодействия с другой системой, выполнять команды (включение/выключение записи, установка пресетов, I/O и т.д.) в канале, получать доступ к архиву. Сведения о данном типе плагина находятся в главе 4.7. Пример плагина представлен проектом EventProcessor.cproj. Если от **MACROSCOP** требуется получать только видео и звук, то существует более простой вариант, рассмотренный в главе 5. Пример получения видео по HTTP представлен проектом HttpVideo.cproj.
3. **Видеоаналитика**
   Любой алгоритм обработки видеопотока может быть реализован с помощью плагина-видеоаналитика. Все результаты работы аналитика представляются событиями, которые могут быть далее интерпретированы плагином-элементом меню или плагином-визуализатором. Данные плагины рассматриваются в главах 4.2, 4.5, 4.6. Пример представлен проектом Analyst.csproj.

3 Процедура регистрации плагинов в MACROSCOP

В момент запуска отдельных компонентов (сервер/клиент/конфигуратор, далее в тексте - *хосм*) программного комплекса **MACROSCOP** происходит поиск .NET сборок в папке Plugins запускаемой программы. В каждой найденной сборке должен быть реализован интерфейс *IPlugin*, представленный ниже:

```
public interface IPlugin
{
        /// <summary>
        /// Возвращает уникальный идентификатор модуля
        /// </summary>
        Guid Id { get; }

        /// <summary>
        /// Возвращает название модуля
        /// </summary>
        string Name { get; }

        /// <summary>
        /// Возвращает название производителя модуля
        /// </summary>
        string Manufacturer { get; }

        /// <summary>
        /// Инициализация модуля.
        /// Вызывается хостом на этапе регистрации модуля в системе.
        /// </summary>
        /// <param name="host">Интерфейс хоста</param>
       void Initialize(IPluginHost host);
 }
```

Пример реализации данного интерфейса:

```
public class ModuleDef : IPlugin
{
    public Guid Id
    {
      get { return new Guid("17EE3457-8FC2-4C0F-B133-EF11D0C4F38C"); }
    }

    public string Name
    {
       get { return "Модуль поиска оставленных предметов"; }
    }

    public string Manufacturer
    {
        get { return "MACROSCOP"; }
    }

    public void Initialize(IPluginHost host)
    {
       host.RegisterAnalyst(typeof(AnalystExample));
       host.RegisterExternalEvent(typeof(ObjectLeftEvent));
    }
}
```

Как только указанный интерфейс был обнаружен хостом, вызывается метод инициализации (*Initialize*), в качестве аргумента которому передается интерфейс *IPluginHost,* предоставляющий сервисные методы хоста:

```
public interface IPluginHost
{
        /// <summary>
        /// Получает интерфейс менеджера протоколов
        /// </summary>
        /// <returns></returns>
        IMcLogMgr GetLogManager();

        /// <summary>
        /// Регистрация устройства.
        /// </summary>
        void RegisterDevType(DevType_RegInfo regInfo);

        /// <summary>
        /// Регистрация получателя кадров.
        /// </summary>
        void RegisterRTFR(RTFR_RegInfo regInfo);

                .
                .
                .

        /// <summary>
        /// Регистрирует внешнее событие
        /// </summary>
        void RegisterExternalEvent(Type eventType);

       /// <summary>
       /// Регистрирует внешнее действие
       /// </summary>
       void RegisterExternalAction(Type actionType);

       /// <summary>
       /// Регистрирует пункт меню в MACROSCOP клиенте
       /// </summary>
       void RegisterMenuItem(Type menuItemType, List<Guid> requiredPluginIDs);

}
```

Сервисные методы хоста предоставляют следующие возможности:

- Протоколирование работы плагинов с помощью интерфейса *IMcLogMgr*, получаемого соответствующим методом *GetLogManager()*.
- Регистрация в системе плагинов для решения различных задач.

4 Плагины

В данной главе подробно рассматривается каждый тип плагина. Наиболее важные фрагменты кода отражены в тексте.

4.1 Плагин-действие

Плагин действие позволяет расширить список функций в сценариях и планировщике задач по расписанию. Для создания такого типа плагина необходимо наследоваться от класса *ExternalAction*:

```
/// <summary>
/// Базовый класс действия.
/// </summary>
[Serializable]
public abstract class ExternalAction : IAction
{
    [NonSerialized]
    protected IActionHost actionHost;

    /// <summary>
    /// Инициализация. Вызывается хостом перед началом работы.
    /// </summary>
    /// <param name="host"></param>
    public virtual void Initialize(IActionHost host)
    {
        actionHost = host;
    }

    /// <summary>
    /// Пользовательский элемент настройки действия. Вызывается конфигуратором.
    /// Должен возвращать тип UserControl (WPF).
    /// </summary>
    public abstract object GetGUISettingsControl();

    /// <summary>
    /// Показывает правильно ли на данный момент
    /// сконфигурировано ли действие
    /// </summary>
    public abstract bool IsConfigurated
    {
        get;
    }

    /// <summary>
    /// Свойство показывает, привязано ли действие к
    /// конкретному каналу. Иными словами, влияет ли действие каким-либо
    /// образом на канал.
    /// </summary>
    public virtual bool IsChannelIndependent
    {
        get
        {
            //по умолчанию действие к каналу никак не привязано
            return true;
        }
    }
```

```
        /// <summary>
        /// Запуск действия на выполнение
        /// </summary>
        public abstract void Run(RawChannelEvent channelEvent);

        /// <summary>
        /// Выполнение команд в канале. Заполняется сервером, может использоваться в
        /// методе Run (см. выше)
        /// </summary>
        [NonSerialized]
        public ExecuteCommandDelegate ExecuteCommand;
    }
```

В классе наследнике требуется задать следующие атрибуты:

- *ActionGUIName* – Название действия, которое будет фигурировать в графическом интерфейсе в конфигураторе.
- *GuidAttribute* – Идентификатор действия.

Опциально может быть задан атрибут *ActionNeedsEventArgument*, который показывает, что для вызова метода *Run* плагина со стороны сервера обязательно следует передавать объект события. Это также означает, что действие выполняется только по событию и не может выполняться в задачах по расписанию, при выполнении которых события не возникают.

Также требуется определить (переопределить) методы базового класса. Рассмотрим более подробно метод инициализации, в который передается интерфейс *IActionHost* со стороны хоста. Интерфейс выглядит следующим образом:

```
/// <summary>
/// Интерфейс, предоставляющий возможности
/// хоста при инициализации плагина действия.
/// </summary>
public interface IActionHost
{
    /// <summary>
    /// Получение информации о канале,
    /// в котором запущен текущий экземпляр
    /// плагина действия.
    /// </summary>
    /// <returns></returns>
    RawChannelInfo GetChannelInfo();

    /// <summary>
    /// Сохраняет любой сериализуемый объект в конфигурацию
    /// MACROSCOP. Используется в конфигураторе.
    /// </summary>
    /// <param name="id">Идентификатор объекта</param>
    /// <param name="obj">Объект</param>
    void SaveObject(Guid id, object obj);

    /// <summary>
    /// Получает ранее из сериализованный объект из конфигурации.
    /// Используется в конфигураторе.
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    object GetObject(Guid id);
 }
```

Данный интерфейс позволяет получить информацию о канале, к которому привязан экземпляр плагина. Эта информация включает в себя имя канала и его идентификатор. Идентификатор должен быть использован при выполнении команд в канале (см. делегат *ExecuteCommand*). Подробную информацию о доступных командах вы можете прочитать в разделе 4.7. Кроме того, данный интерфейс с помощью методов SaveObject и GetObject позволяет сохранять и загружать любой сериализуемый объект по идентификатору на этапе задания пользователем конфигурации **MACROSCOP**. Такой механизм позволяет сохранять и извлекать настройки, которые являются одинаковыми для всех экземпляров плагинов.

Метод *GetGUISettingsControl* должен возвращать элемент типа UserControl, содержащий графический интерфейс для настройки данного экземпляра плагина в конфигураторе. Весь графический интерфейс должен быть выполнен с использованием WPF (Windows Presentation Foundation). Свойство IsConfigurated показывает правильно ли пользователь сконфигурировал действие в графическом интерфейсе. Если неправильно, то конфигурацию невозможно будет применить до тех пор, пока ошибки не будут исправлены.

Для регистрации плагина-действия в системе, необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс *IPlugin,* вызвать метод *RegisterExternalAction* интерфейса хоста (см. также 0).

4.2 Плагин-видеоаналитик

Данный тип плагина осуществляет покадровый анализ видеопотока, что позволяет создавать сервисные детекторы, например, детектор оставленных предметов, детектор саботажа и др. Для создания такого типа плагина, необходимо создать наследника базового класса *VideoAnalyst*:

```
/// <summary>
/// Класс видеоаналитика. Используется для обработки кадров и карт движения.
/// </summary>
public abstract class VideoAnalyst : IDisposable
{
     /// <summary>
     /// Инициализация аналитика. Вызывается хостом перед началом процесса обработки.
     /// </summary>
     /// <param name="mdZones">Зоны детектирования движения.</param>
     /// <param name="settings">Настройки аналитика.</param>
     public abstract void Initialize(List<MDZone> mdZones, PluginSettings settings);

      /// <summary>
      /// Метод обработки кадра и карты движения. Вызывается хостом.
      /// </summary>
      /// <param name="frameDT">Временная метка кадра</param>
      /// <param name="width">Ширина кадра</param>
      /// <param name="height">Высота кадра</param>
      /// <param name="offset">Смещение в массиве пикселей</param>
      /// <param name="stride">Размер строки кадра в байтах</param>
      /// <param name="pixels">Массив пикселей</param>
      /// <param name="bpp">Бит на пиксель</param>
      /// <param name="motionMap">Карта движения, может быть null</param>
      public abstract void Process(DateTime frameDT, int width, int height, int offset,
                        int stride, byte[] pixels, int bpp, MotionMap motionMap);

      /// <summary>
      /// Генерирует в канале ранее зарегистрированное внешнее событие. Заполняется
```

```
        /// хостом. Вызывается аналитиком.
        /// </summary>
        public GenerateEventDelegate GenerateEvent;

        /// <summary>
        /// Поддерживает ли аналитик работу с частично продекодированными кадрами.
        /// True означает, что на вход могут подаваться уменьшенные видеокадры,
        /// False означает, что на вход всегда будут подаваться видеокадры с оригинальным
        ///разрешением.
        /// </summary>
        public virtual bool SupportsPartlyDecodedFrames
        {
            get
            {
               return false;
            }
        }

        /// <summary>
        /// Поддерживаемый формат пикселя
        /// </summary>
        public virtual VAPixelFormat PixelFormat
        {
            get
            {
                return VAPixelFormat.BGR24;
            }
        }

        /// <summary>
        /// Выполняет команду.
        /// </summary>
        /// <param name="cmdObj"></param>
        /// <returns></returns>
        public abstract object ProcessCommand(object cmdObj);

        /// <summary>
        /// Освобождение ресурсов
        /// </summary>
        public abstract void Dispose();

        /// <summary>
        /// Изменяет общие или специфические настройки аналитика, если это необходимо.
        /// Вызов метода должен приводить к появлению пользовательского окна настройки.
        /// Вызывается хостом (конфигуратором).
        /// </summary>
        /// <param name="settingsHost"></param>
        /// <param name="settings">Текущие параметры аналитика</param>
        /// <returns>Новые параметры аналитика</returns>
        public abstract PluginSettings SetSettings(ISettingsHost settingsHost,
                                                   PluginSettings settings);
    }
```

Класс наследник должен переопределить все абстрактные методы базового класса и, при необходимости, виртуальные свойства. Кроме того, производный класс должен иметь обязательный атрибут *PluginGUINameAttribute*, содержащего название аналитика, отображаемого в графической оболочке конфигуратора **MACROSCOP**. Если аналитик может быть настроен в конфигураторе, необходимо реализовать метод настройки *SetSettings*, а также указать атрибут *PluginHasSettingsAttribute*.

Поскольку каждый аналитический плагин прикрепляется пользователем к тому или иному каналу в конфигураторе **MACROSCOP**, объект настроек *PluginSettings* состоит из 2 частей:

1) Настройки, связанные с текущим каналом безопасности
2) Общие настройки аналитика, не зависящие от выбранного канала безопасности

```
public struct PluginSettings
{
   /// <summary>
   /// Специфические настройки плагина, связанные с текущим каналом. Может быть null.
   /// Объект настроек должен быть сериализуемым.
   /// </summary>
   public object channelSpecificSettings;

   /// <summary>
   /// Общие настройки плагина. Не зависят от текущего канала. Может быть null.
   /// Объект настроек должен быть сериализуемым.
   /// </summary>
   public object generalSettings;
 }
```

Если настройки аналитика едины для всех каналов, то достаточно заполнять только объект общих настроек. В противоположном случае, когда все настройки аналитика привязываются к конкретному каналу, достаточно заполнять только объект специфических настроек канала. Объекты общих и специфических настроек являются пользовательскими. Единственным требованием для них является возможность их сериализации, поскольку все настройки аналитиков хранятся в общей конфигурации **MACROSCOP**. Рассмотрим также метод *ProcessCommand*, который позволяет аналитику выполнять команды от плагина-элемент меню (см. 4.7). Данный метод получает команду в виде сериализуемого объекта, который формируется на стороне плагина-элемент меню. В качестве результата данный метод также должен возвращать сериализуемый объект, который впоследствии получит и обработает плагин-элемент меню. Такой механизм позволяет реализовать клиент-серверное взаимодействие, поскольку плагин видеоаналитик работает на сервере, а плагин-элемент меню всегда в клиенте.

В процессе обработки видеокадров аналитик должен передавать результаты своего анализа серверу с помощью генерации событий. Для этого необходимо заранее (см. 0) зарегистрировать на стороне хоста внешнее пользовательское событие, содержащее описание всех необходимых полей для интерпретации результатов работы аналитика. Регистрируется событие с помощью метода *RegisterExternalEvent* интерфейса *IPluginHost* на этапе инициализации модуля. Пользовательское событие должно быть унаследовано от базового класса *RawChannelEvent*:

```
    /// <summary>
    /// Родительский класс в иерархии событий, происходящих в канале.
    /// </summary>
    [Serializable]
    public abstract class RawChannelEvent
    {
        /// <summary>
        /// ВременнАя метка события.
        /// </summary>
        public DateTime EventTime;

        /// <summary>
```

```
        /// Комментарий к событию.
        /// </summary>
        public string Comment;



        /// <summary>
        /// Является ли событие локальным.
        /// Локальные события не отправляются за пределы текущего хоста.
        /// </summary>
        public bool IsLocal = false;

        /// <summary>
        /// Режим сохранения события в БД
        /// </summary>
        public abstract EventArchiveSaveMode SaveMode
        {
            get;
        }

        public RawChannelEvent()
        {
            EventTime = DateTime.UtcNow;
        }
    }
```

Каждое пользовательское событие должно иметь ряд обязательных атрибутов:

1) *GuidAttribute* – определяет в явном виде уникальный идентификатор события
2) *Serializable* – позволяет выполнять сериализацию события

Кроме того, имеется ряд необязательных (опциональных) атрибутов:

1) *EventNameGUI* – задает название события в графическом интерфейсе хоста. Если атрибут не задан, то событие не будет отображаться в конфигураторе в разделе сценариев
2) *EventNameDatabase* – название таблицы в базе данных, в которую будут сохраняться экземпляры события; атрибут должен использоваться в случае, если у события есть поля, которые должны быть сохранены в БД (важно отметить, что свойство события *SaveMode* должно принимать в этом случае значения *Special* или *Both*)
3) *EventGeneratesAlarmByDefault* – событие по умолчанию является тревожным, что означает автоматическую привязку действия “Генерировать тревогу” к событию при создании нового канала в конфигураторе **MACROSCOP**
   EventGenerationFrequency – указывает на частоту генерации данного события, требует параметр *EventGenerationFrequencyMode* (см. ниже). Данный атрибут влияет на работу сервера при сохранении событий в БД и имеет рекомендательный характер. Если атрибут не был указан, то по умолчанию используется режим *Low*. Этот режим является предпочтительным.

```
        /// <summary>
        /// Частота генерации события
        /// </summary>
        public enum EventGenerationFrequencyMode
        {
            /// <summary>
            /// События генерируется с частотой,
            /// близкой к частоте анализа кадров или более
            /// </summary>
            High = 0,
```

```
        /// <summary>
        /// События генерируются с частотой, сопоставимой
        /// с половиной частоты анализа кадров
        /// </summary>
        Middle,


        /// <summary>
        /// События генерируются значительно реже
        /// частоты анализа кадров
        /// </summary>
        Low
    }
```

Если пользовательское событие имеет поля, которые должны сохраняться в БД, необходимо для каждого из них указывать атрибут *EventFieldSaveable.* В качестве параметров атрибут требует указать порядковый номер поля *Order* (отсчет начинается с 0) и флаг *IsIndexable*, указывающий, нужно ли индексировать данное поле.

Поддерживаются следующие типы полей события, к которым применимы указанные выше атрибуты: **int**, **bool**, **long**, **double**, **DateTime**, **string**, **Guid**, **byte[]**.

Свойство *SaveMode* определяет, будет ли событие сохраняться в БД. Событие может храниться как в базовой таблице, в которой находятся только поля класса *RawChannelEvent*, так и в специальной, содержащей все поля события. Также имеется возможность сохранять событие в обе таблицы. Использование базовой таблицы позволяет фиксировать сам факт возникновения события в системе. В будущем пользователь может читать из базовой таблицы события штатными средствами **MACROSCOP**.

Таким образом, пользовательское событие может быть представлено следующим образом:

```
[GuidAttribute("389EDCE2-54BB-4C2C-9984-51B7516A5DDF")]
[EventNameGUI("Оставлен предмет")]
[EventDatabaseName("objectleft")]
[EventGeneratesAlarmByDefault]
[EventGenerationFrequency(EventGenerationFrequencyMode.Low)]
[Serializable]
public class ObjectLeftEvent : RawChannelEvent
{
    [EventFieldSaveable(0, true)]
    [EventFieldNameGUI("Предмет")]
    private string objectName;

    public ObjectLeftEvent(string objectName)
    {
        this.objectName = objectName;
    }

    public override EventArchiveSaveMode SaveMode
    {
        get  { return EventArchiveSaveMode.Both; }
    }
}
```

Для регистрации плагина-видеоаналитика в системе, необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс *IPlugin,* вызвать метод *RegisterAnalyst* интерфейса хоста (см. также 0).

4.3 Плагин-детектор движения

Плагин данного типа позволяет сделать альтернативный детектор движения для **MACROSCOP**. Для этого необходимо создать наследника класса *MotionDetector:*

```
/// <summary>
/// Детектор движения.
/// Базовый абстрактный класс для всех детекторов движения.
/// </summary>
public abstract class MotionDetector
{
	/// <summary>
	/// Зоны детектирования.
	/// </summary>
	protected List<MDZone> zones = new List<MDZone>();

	/// <summary>
	/// Добавляет 1 зону с маской, занимающей весь кадр.
	/// </summary>
	public void AddFullFrameZone()
	{
		MDZone zone = new MDZone();
		zones.Add(zone);
	}

	/// <summary>
	/// Добавляет 1 зону с маской, занимающей весь кадр, с указанием параметров
	/// зоны.
	/// </summary>
	public void AddFullFrameZone(float minObjWidth, float minObjHeight)
	{
		MDZone zone = new MDZone(0, minObjWidth, minObjHeight);
		zones.Add(zone);
	}

	/// <summary>
	/// Добавление зоны.
	/// </summary>
	/// <param name="zone">Добавляемая зона.</param>
	public void AddZone(MDZone zone)
	{
		zones.Add(zone);
	}

	/// <summary>
	/// Количество зон. Только чтение.
	/// </summary>
	public int ZonesCount
	{
		get
		{
			return zones.Count;
		}
	}

	#region Параметры необходимые для штатного детектора движения
		…
```

```
        #endregion

        /// <summary>
        /// Детектирование движения в заданном кадре.
        /// </summary>
        /// <param name="width">Ширина кадра.</param>
        /// <param name="height">Высота кадра.</param>
        /// <param name="bgr24bytes">Массив байт в формате bgr24, составляющих
        ///кадр.
        /// </param>
        /// <param name="timestamp">Временная метка.</param>
        /// <returns>Возвращает карту движения. Индекс > 0 означает наличие
        /// движения в рассматриваемом пикселе. Значение индекса в карте движения
        /// соответствует индексу движущегося объекта.
        ///</returns>
        public abstract MotionMap Detect(int width, int height, int offset, int
                                         stride, byte[] bgr24bytes, DateTime timestamp);
    }
```

Поле *zones* заполняется сервером перед началом работы данного типа плагина через методы *AddFullFrameZone* и *AddZone*. Все поля в регионе “Параметры необходимые для штатного детектора движения” используются только встроенным детектором движения и должны быть оставлены без изменений. Основная логика работы плагина концентрируется в методе *Detect*, на вход которому подается кадр в формате bgr24 (каналы Blue, Green, Red, 8 бит на каждый канал). Результатом данного метода должна быть карта движения *MotionMap*. Для создания карты движения требуется два массива:

1. Карта наличия движения как такового. Представляет собой двумерный массив целых чисел. Каждое число массива отличное от нуля (индекс) идентифицирует движущийся объект. 0 говорит об отсутствии движения.
2. Массив движущихся объектов. Доступ к элементам массива осуществляется по индексам карты движения.

Для регистрации плагина-детектора движения в системе, необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс *IPlugin,* вызвать метод *RegisterMotionDetector* интерфейса хоста (см. также 0).

4.4 Плагин-трекер

Плагин данного типа позволяет сделать трэкер для **MACROSCOP**. Для этого необходимо создать наследника класса *Tracker*:

```
/// <summary>
/// Базовый класс для всех трэкеров.
/// </summary>
public abstract class Tracker
{
        /// <summary>
        /// Обработка кадра и карты движения.
        /// Результатом обработки является измененная карта движения, на которой
        /// корректно проставлены индексы движущихся объектов.
        /// </summary>
        /// <param name="width">Ширина кадра.</param>
        /// <param name="height">Высота кадра.</param>
        /// <param name="offset">Отступв в байтах до начала кадра в массиве
        ///байт.</param>
        /// <param name="stride">Страйд.</param>
        /// <param name="bgr24bytes">Байты с пикселами кадра.</param>
        /// <param name="timestamp">Временная метка.</param>
```

```
        /// <param name="detectedMap">Продетектированная карта движения. Значение
        ///может измениться после обработки.</param>
        public abstract void Process(int width, int height, int offset, int stride,
              byte[] bgr24bytes, DateTime timestamp, ref MotionMap detectedMap);
        /// <summary>
        /// Сбрасывает внутреннее состояние трекера.
        /// </summary>
        public abstract void ResetInternalState();
}
```

Основная логика работы реализуется методом *Process*. Данному методу на вход поступает кадр формата bgr24 (каналы Blue, Green, Red, 8 бит на каждый канал) и карта движения *MotionMap*, полученная плагином-детектором движения. Результатом работы плагина являются идентификаторы сопровождаемых объектов, которые должны быть проставлены в карте движения. Метод *ResetInternalState* должен сбрасывать массив сопровождаемых в данный момент объектов приводя, таким образом, трэкер в начальное состояние.

4.5 Плагин-визуализатор

Плагин данного типа позволяет графически отображать информацию (например, рамки движущихся объектов, распознанные номера, лица и др.) содержащуюся в событиях, которые поступают в каналы клиента **MACROSCOP**. Для создания данного типа плагина необходимо создать наследника класса *RTVisualiser*:

```
/// <summary>
/// Класс визуализатора.
/// </summary>
public abstract class RTVisualiser
{
        /// <summary>
        /// Панель для отрисовки примитивов, текста и другой информации на канале.
        /// </summary>
        protected IDrawingPanel drawingPanel;

        /// <summary>
        /// Контейнер графических элементов. Позволяет размещать отдельные
        /// UserControl'ы в канале.
        /// </summary>
        protected Panel controlsContrainer;
        protected bool showRects;

        /// <summary>
        /// Инициализация визуализатора. Вызывается хостом.
        /// </summary>
        /// <param name="drawingPanel">Панель для рисования.</param>
        /// <param name="controlsContrainer"></param>
        public virtual void Initialize(IDrawingPanel drawingPanel, Panel
                                                    controlsContrainer)
        {
              this.drawingPanel = drawingPanel;
              this.controlsContrainer = controlsContrainer;
        }

        /// <summary>
        /// Обработка события визуалайзером. Специфическая отрисовка результатов
        /// события.
        /// </summary>
        /// <param name="channelId">Идентификатор канала</param>
        /// <param name="chEv">Событие.</param>
```

```
        /// <param name="isAlarm">Является ли событие тревожным</param>
        public abstract void ProcessEvent(Guid channelId, RawChannelEvent chEv,
                                                          bool isAlarm);


        /// <summary>
        /// Включено ли пользователем отображение рамок в данном канале.
        /// </summary>
        public bool ShowRects
        {
               get { return showRects; }
               set { showRects = value; }
        }

        /// <summary>
        /// Делегат для регистрации подпункта во всплывающем меню канала в клиенте
        /// MACROSCOP. Заполняется хостом. Вызывается визуализатором.
        /// </summary>
        /// <param name="item"></param>
        public RegisterChannelMenuItemHandler RegisterChannelMenuItem;


        /// <summary>
        /// Очистка всего, что нарисовано визуализатором.
        /// </summary>
        public abstract void Clear();

        /// <summary>
        /// Освобождение всех ресурсов. В перегружаемых метода в наследниках
        /// обязательно вызывать Release базового класса.
        /// </summary>
        public virtual void Release()
        {
               drawingPanel = null;
               controlsContrainer = null;
        }
}
```

Каждый плагин-визуализатор должен определять метод *ProcessEvent*, на вход которому передается очередное событие, поступившее в канал. Все события генерируются **MACROSCOP** или другими плагинами. Плагин может визуализировать любые типы событий, а их фильтрацию можно осуществить с помощью оператора if и ключевого слова is, например:

```
        if (chEv is CounterEvent)
        {
               ...
        }
```

Визуализатор может регистрировать свой подпункт во всплывающем меню, отображаемом при щелчке правой кнопки мыши на канале в клиенте (см. Рис. 1.).
Это позволяет изменять логику работы визуализатора в зависимости от предпочтений пользователя. Данная возможность предоставляется делегатом *RegisterChannelMenuItem*.

Для регистрации плагина-визуализатора в системе, необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс *IPlugin,* вызвать метод *RegisterRTVisualiser* интерфейса хоста (см. также 0).

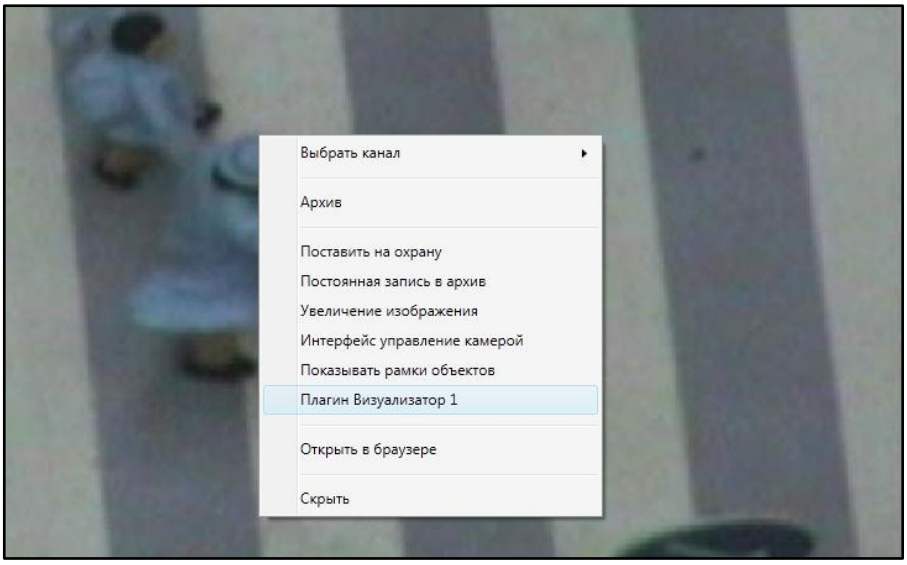

Рис. 1 Меню по нажатию правой клавиши мыши в канале

4.6 Плагин-элемент меню

Плагин данного типа позволяет создавать графический интерфейс в клиенте **MACROSCOP**, который может быть вызван пользователем при щелчке по соответствующему пункту меню кнопки «Настройка» (см. Рис.2).

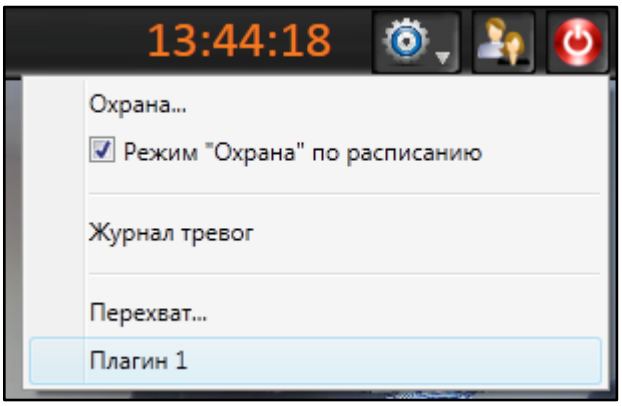

Рис. 2 Меню по нажатию на кнопки «Настройка»

Типичное применение данного плагина заключается в организации клиент-серверного взаимодействия с другими плагинами. Например, плагин может обрабатывать результаты работы аналитических плагинов, работающих на сервере. Для создания плагина-элемента меню необходимо создать наследника класса *ClientMenuItem*:

```
/// <summary>
/// Класс элемент меню. Используется в клиенте для добавления
/// подпунтка в меню кнопки "Настройка".
/// </summary>
public abstract class ClientMenuItem : IDisposable
{
    private bool isEnabled = true;
```

```
        /// <summary>
        /// Инициализация плагина.
        /// </summary>
        /// <param name="toolSet"></param>
        public abstract void Initialize(IPluginToolSet toolSet);

         /// <summary>
         /// Название элемента меню.
         /// </summary>
        public abstract string Name
        {
            get;
        }

         /// <summary>
         /// Включен ли в данный момент элемент меню.
         /// </summary>
        public bool IsEnabled
        {
            get
            {
                return isEnabled;
            }
            set
            {
                isEnabled = value;
            }
        }

        /// <summary>
        /// Метод обработки щелчка мыши (нажатия клавиатуры)
        /// </summary>
        public abstract void OnClicked();


        /// <summary>
        /// Метод освобождения ресурсов.
        /// </summary>
        public abstract void Dispose();
    }
```

В классе-наследнике необходимо определить метод инициализации, в который со стороны хоста (клиента **MACROSCOP**) передается интерфейс с сервисными функциями. Эти функции позволяют получить идентификаторы каналов и их имена в текущей конфигурации. Кроме того, они предоставляют доступ к архиву **MACROSCOP**, возможность отправлять команды аналитическим плагинам на сервере и получать результат их исполнения. Имеется возможность подписываться на любые события, возникающие в системе. Интерфейс представляется следующим образом:

```
    /// <summary>
    /// Интерфейс предоставляемый хостом для
    /// доступа в архив, для отправки команд, для
    /// подписки на события в системе.
    /// </summary>
     public interface IPluginToolSet
     {
        /// <summary>
        /// Получает интерфейс для работы с архивом
        /// </summary>
        /// <returns></returns>
        IArchiveEventsReader GetArchiveReader();
```

```
    /// <summary>
    /// Отправляет команду плагину,
    /// работающего на сервере.
    /// </summary>
    /// <param name="pluginId">Идентификатор плагина</param>
    /// <param name="channelsId">Идентификаторы каналов</param>
    /// <param name="cmdObj">Команда</param>
    /// <returns>Возвращает результат от каждого канала. Ключ - идентификатор канала.
    /// Значение - результат выполнения команды.</returns>
     Dictionary<Guid, object> SendChannelsCommand(Guid pluginId, List<Guid>
                                                  channelsId, object cmdObj);

    /// <summary>
    /// Устанавливает/удаляет обработчик событий, возникающих в канале.
    /// </summary>
    /// <param name="subscrId">Идентификатор подписчика</param>
    /// <param name="channelId">Идентификатор канала</param>
    /// <param name="eventsHandler">Обработчик. Если null, то ранее установленный
                                                  обработчик удаляется.</param>
    void SetEventsHandler(Guid subscrId, Guid channelId, EventsHandler eventsHandler);
}
```

Метод *GetArchiveReader* предоставляет интерфейс доступа к архиву и к информации о текущих каналах в конфигурации. Подробные сведение о данном интерфейсе вы можете найти в исходных кодах **MACROSCOP SDK**. Метод *SendChannelsCommand* отправляет команду разным экземплярам плагина одного и того же типа по указанному списку каналов и получает результаты исполнения команды. Метод SetEventsHandler устанавливает или удаляет обработчик событий на заданном канале.

В методе *OnClicked* должна быть реализована логика работы с пользователем через графический интерфейс. Данный метод вызывается клиентом в момент щелчка клавиши мыши (клавиатуры) по пункту меню, связанному с плагином.

Для регистрации плагина-элемента меню, необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс *IPlugin,* вызвать метод *RegisterMenuItem* интерфейса хоста (см. также 0).

4.7 Плагин-процессор событий

Процессор событий позволяет получать сигналы от сторонних систем и обрабатывать их. В процессе обработки сигналов данный тип плагина может, как выполнять команды в канале, в котором он существует, так и генерировать в этом канале события. Плагин-процессор событий создается путем наследования от базового класса *EventProcessor*:

```
/// <summary>
/// Класс плагина для обработки событий системы, генерации команд и своих событий
/// </summary>
public abstract class EventProcessor
{
    /// <summary>
    /// Инициализация. Вызывается хостом.
    /// </summary>
    /// <param name="archiveReader">Интерфейс доступа к архиву</param>
    /// <param name="channelSpecificSettings">Настройки плагина</param>
   public abstract void Initialize(IArchiveEventsReader archiveReader,
                                                    PluginSettings settings);

    /// <summary>
    /// Генерирует в каналее ранее зарегистрированное внешнее событие. Заполняется
```

```
        /// хостом. Вызывается плагином.
        /// </summary>
        public GenerateEventDelegate GenerateEvent;

        /// <summary>
        /// Запускает заданную команду на выполнение в канале. Заполняется хостом.
        /// Вызывается плагином.
        /// </summary>
        public ExecuteCommandExDelegate ExecuteCommand;

        /// <summary>
        /// Позволяет подписываться на события, происходящие в канале. Заполняется
        /// плагином при необходимости на этапе инициализации.
        /// </summary>
        public ReceiveEventDelegate OnChannelEventReceived;

        /// <summary>
        /// Изменяет общие или специфические настройки, если это необходимо.
        /// Вызов метода должен приводить к появлению пользовательского окна настройки.
        /// Вызывается хостом (конфигуратором).
        /// </summary>
        /// <param name="settings">Текущие настройки плагина.</param>
        /// <returns>Новые настройки плагина.</returns>
        public abstract PluginSettings SetSettings(PluginSettings settings);
    }
```

Производный класс должен иметь обязательный атрибут *PluginGUINameAttribute*, содержащего название аналитика, отображаемого в графической оболочке конфигуратора **MACROSCOP**. Если аналитик может быть настроен в конфигураторе, необходимо реализовать метод настройки *SetSettings*, а также указать атрибут *PluginHasSettingsAttribute*.

В метод инициализации *Initialize* со стороны хоста передается интерфейс для доступа в архив **MACROSCOP** и объект настроек плагина. Поскольку каждый плагин прикрепляется пользователем к тому или иному каналу в конфигураторе **MACROSCOP**, объект настроек *PluginSettings* состоит из 2 частей:

1) Настройки, связанные с текущим каналом безопасности
2) Общие настройки аналитика, не зависящие от выбранного канала безопасности

```
public struct PluginSettings
{
   /// <summary>
   /// Специфические настройки плагина, связанные с текущим каналом. Может быть null.
   /// Объект настроек должен быть сериализуемым.
   /// </summary>
   public object channelSpecificSettings;

   /// <summary>
   /// Общие настройки плагина. Не зависят от текущего канала. Может быть null.
   /// Объект настроек должен быть сериализуемым.
   /// </summary>
   public object generalSettings;
 }
```

Если настройки едины для всех каналов, то достаточно заполнять только объект общих настроек. В противоположном случае, когда все настройки плагина привязываются к конкретному каналу, достаточно заполнять только объект специфических настроек канала. Объекты общих и специфических настроек являются пользовательскими.

Единственным требованием для них является возможность их сериализации, поскольку все настройки процессоров событий хранятся в общей конфигурации **MACROSCOP**.

Если процессору событий в качестве результата своей работы необходимо выполнить определенную команду в канале (например, включить/выключить запись, установить пресет на камере, повернуть камеру и т.д.), требуется создать объект команды. На данный момент существуют следующие команды:

- *RawEnableRecordingCommand* – включает запись в канале, опционально указывается интервал записи
- *RawDisableRecordingCommand* – выключает запись в канале
- *RawGoToPresetPtzCommand* – устанавливает пресет на камере
- *RawGoHomePtzCommand* – устанавливает домашнее положение камеры
- *RawStopPtzCommand* – останавливает выполнение ptz команд на камере
- *RawMovePtzCommand* – перемещение на шаг
- *RawZoomPtzCommand* – относительное приближение (зум)
- *RawStartMovePtzCommand* – непрерывное (желательно плавное) движение
- *RawMoveToPtzCommand* - поворачивает камеру таким образом, что указанная точка оказывается в центре области кадра
- *RawSetOutputIOCommand* – устанавливает уровень сигнала на выходе камеры
- *RawSetOutputPulsesIOCommand* – генерирует последовательность импульсов на выходе камеры

Плагин может подписываться на события, возникающие в канале. Для этого на этапе своей инициализации процессор событий должен заполнить делегат *OnChannelEventReceived*. Кроме того, плагин может генерировать свои события в канале. Подробно о регистрации событий вы можете прочитать в разделе 4.2.

Для регистрации плагина-процессор событий, необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс *IPlugin,* вызвать метод *RegisterEventProcessor* интерфейса хоста (см. также 0).

4.8 Плагин-получатель кадров

Плагин данного типа позволяет получать видео и звук с IP устройств, данные о детекции движения, управлять поворотными камерами, управлять входами/выходами (IO) устройства. Для решения подзадачи получения кадров необходимо реализовать интерфейс *IRealTimeFrameReceiver*:

```
/// <summary>
/// Интерфейс получения кадров реального времени.
/// Используется для создания плагинов, получающих кадры
/// с IP-камер.
/// </summary>
public interface IRealTimeFrameReceiver
{
        /// <summary>
        /// Обработчик события о приходе нового кадра.
        /// Вызывается реализующей интерфейс стороной.
        /// На событие подписывается хост.
        /// </summary>
        event NewRawFrameEventHandler NewRawFrame;

        /// <summary>
        /// Обработчик событий. Вызывается реализующей интерфейс стороной.
```

```
        /// На обработчик подписывается хост.
        /// </summary>
        event NewRawEventHandler NewEvent;

        /// <summary>
        /// Обработчик добавления записей в лог. Уведомляет хоста, о том, что
        /// поле ConnectionLog (см. ниже) изменилось.
        /// Лог просматривается в конфигураторе при нажатии на кнопку "Журнал"
        /// Вызывается реализующей интерфейс стороной.
        /// На обработчик подписывается хост.
        /// </summary>
        event EventHandler NewLogRecord;

        /// <summary>
        /// Флаг, указывающий нужно ли писать в лог.
        /// Изменяется хостом.
        /// </summary>
        bool IsWritingConnectionLog
        {
            get;
            set;
        }

        /// <summary>
        /// Текущее содержимое лога подключения.
        /// </summary>
        string ConnectionLog
        {
            get;
        }

        /// <summary>
        /// Является ли поток активным
        /// </summary>
        /// <param name="streamType">Тип потока</param>
        /// <returns></returns>
        bool IsStreamActive(ChannelStreamTypes streamType);

        /// <summary>
        /// Запускает поток указанного типа для получения кадров
        /// </summary>
        /// <param name="streamType"></param>
        void StartStream(ChannelStreamTypes streamType);

        /// <summary>
        /// Останавливает поток указаного типа
        /// </summary>
        /// <param name="streamType"></param>
        void StopStream(ChannelStreamTypes streamType);

        /// <summary>
        /// Отправляет звук на устройство (случай реализации дуплексного звука).
        /// </summary>
        /// <param name="soundData"></param>
        void SendSound(byte[] soundData);

        /// <summary>
        /// Освобождает все ресурсы. Закрывает все потоки.
        /// </summary>
        void Release();
    }
```

При реализации данного интерфейса необходимо создать механизм работы с потоками данных. Потоки данных представляют собой или последовательность кадров определенного типа, или последовательность событий. В текущей версии **MACROSCOP SDK** имеются следующие типы потоков данных:

```
/// <summary>
/// Типы потоков канала.
/// </summary>
public enum ChannelStreamTypes : ulong
{
        /// <summary>
        /// Главный поток видео
        /// </summary>
        MainVideo = 1,

        /// <summary>
        /// Альтернативный поток видео
        /// </summary>
        AlternativeVideo = 2,

        /// <summary>
        /// Поток звука, идущий с камеры.
        /// </summary>
        MainSound = 4,

        /// <summary>
        /// Альтернативный звуковой поток
        /// </summary>
        AlternativeSound = 8,

        /// <summary>
        /// Обратный поток звука.
        /// </summary>
        OutputSound = 16,

        /// <summary>
        /// Поток данных детекции движения.
        /// </summary>
        MotionDetection = 32,

        /// <summary>
        /// Поток данных от системы ввода-вывода камеры
        /// </summary>
        IO = 64,
}
```

В зависимости от возможностей подключаемого устройства, а также от настроек канала в конфигураторе, необходимо генерировать те или иные потоки данных. Потоки данных *MainVideo* и *AlterntaiveVideo* состоят из видеокадров *RawVideoFrame*, получаемых с IP устройства. Потоки данных *MainSound* и *AlternativeSound*, *OutputSound* состоят из последовательности звуковых кадров *RawSoundFrame*. Поток данных *MotionDetection* – из последовательности событий *RawChEv_MDresults* и *RawChEv_NoDetection*. Поток IO – из событий *RawChEv_InputSignalLevelChanged*.

Запуск потоков данных происходит со стороны хоста с помощью метода StartStream, в котором производится необходимая инициализация очередного потока. Метод должен сразу же возвращать управление хосту, а длительные операции (операции ввода/вывода) выполнять в отдельных потоках. Остановка потоков и проверка их активности методами StopStream и IsStreamActive также вызывается со стороны хоста и должны выполняться

немедленно без выполнения долгосрочных операций. Результаты своей работы потоки должны возвращать хосту через вызовы обработчиков кадров (NewRawFrame) и событий (NewEvent). Например, если IP-устройство шлет MJPEG кадры, то поток данных *MainVideo* (или AlternativeVideo) должен вызывать NewRawFrame и в качестве одного из аргументов передавать видеокадр *RawMJPEGFrame*:

```
/// <summary>
/// MJPEG кадр
/// </summary>
[Serializable]
public class RawMJPEGFrame : RawVideoFrame
{
 public RawMJPEGFrame()
 {
 }

 /// <summary>
 /// Данные кадра
 /// </summary>
 /// <param name="data"></param>
 public RawMJPEGFrame(byte[] data)
 {
     Data = data;
 }
}
```

Аналогично для потока данных *MainSound* (или *AlterntaiveSound*) и звука в формате G.711U, необходимо передавать кадр *RawG711UFrame*:

```
/// <summary>
/// Кадр стандарта G.711U
/// </summary>
[Serializable]
public class RawG711UFrame : RawSoundFrame
{

    /// <summary>
    /// Данные кадра
    /// </summary>
    /// <param name="data"></param>
    public RawG711UFrame(byte[] data)
    {
        this.Data = data;
        this.samplesRate = 8000;
        this.bitsPerSample = 16;
        this.channels = 1;
        this.bitrate = 64000;
    }
}
```

После создания кадра подходящего типа, требуется обязательно заполнить следующие поля базового класса *RawFrame*:

1. *Id* – идентификатор кадра, состоящий из двух частей. Первая часть – идентификатор последовательности. Идентификатор последовательности генерируется случайным образом перед началом работы потока получения данных. Вторая часть – порядковый номер кадра.
2. *Timestamp* – временная метка кадра, задается в формате UTC.

В форматах .MPEG4 и .H264 для P-кадров обязательно требуется заполнять поле *Dependencies*, содержащее идентификаторы все кадров, от которых данный кадр зависит. В каждом I-кадре требуется указывать инициализирующую информацию для декодера в поле *SpecInitData*.
Пример создания MJPEG-кадра:

```
RawMJPEGFrame jpegFrame = new RawMJPEGFrame(frameData);
jpegFrame.Id.SeqId = videoSeqId;
jpegFrame.Id.NumInSeq = videoNumInSeq++;
jpegFrame.TimeStamp = DateTime.UtcNow;
```

Где изначально были заданы:

```
//Идентификатор последовательности видео кадров
Guid videoSeqId = Guid.NewGuid();
//Номер очередного видео кадра в последовательности
long videoNumInSeq = 0;
```

Пример H.264 I-кадра:

```
RawH264_I_Frame iFrame = new RawH264_I_Frame(frameData);
iFrame.Id.SeqId = videoSeqId;
iFrame.Id.NumInSeq = videoNumInSeq++;
iFrame.TimeStamp = DateTime.UtcNow;
iFrame.SpecInitData = initData;
```

Где initData – инициализирующая информация для декодра.
Пример H.264 P-кадра:

```
RawH264_P_Frame pFrame = new RawH264_P_Frame(frameData);
pFrame.Id.SeqId = videoSeqId;
pFrame.Id.NumInSeq = videoNumInSeq++;
pFrame.TimeStamp = DateTime.UtcNow;
pFrame.Dependencies = frameDependencies;
```

Где frameDependencies – массив идентификаторов кадров, от которых зависит данный кадр. Иными словами совокупность кадров, которая должна быть продекодирована перед декодированием данного кадра.
Пример G.711U кадра:

```
RawG711UFrame g711Frame = new RawG711UFrame(frameData);
g711Frame.Id.SeqId = audioSeqId;
g711Frame.Id.NumInSeq = audioNumInSeq++;
g711Frame.TimeStamp = DateTime.UtcNow;
```

В случае если в процессе получения потоков данных произошел обрыв связи с IP-устройством, плагин-получатель кадров должен уведомить об этом хоста, путем генерации события *RawChEv_NoDataConnection* с обязательно заполненным полем *StreamTypes*, показывающее в каких именно потоках данных произошел обрыв соединения.

Для регистрации получателя кадров в системе, необходимо заполнить регистрационную информацию устройства *DevType_RegInfo* и регистрационную информацию плагина-получателя кадров *RTFR_RegInfo*. Ниже приведено описание класса *DevType_RegInfo*:

```
/// <summary>
/// Регистрационная информация устройства.
/// используется при регистрации плагина,
/// получающего кадры.
/// </summary>
public class DevType_RegInfo
{
        /// <summary>
        /// Идентификатор устройства.
        /// </summary>
        public Guid DeviceTypeGuid;

        /// <summary>
        /// Имя производителя.
        /// </summary>
        public string DevTypeBrandName;

        /// <summary>
        /// Имя устройства.
        /// </summary>
        public string DevTypeModelName;

        /// <summary>
        /// Список возможностей устройства.
        /// </summary>
        public DevType_Capabilities Capabilities;

        /// <summary>
        /// Список доступных разрешений для данного устройства.
        /// </summary>
        public List<VideoResolutions> AvailableResolutions = new
                                                List<VideoResolutions>();

        /// <summary>
        /// Делегат, позволяющий хосту изменять настройки на камере/видеосервере.
        /// </summary>
        public SetDeviceParametersDelegate SetDeviceParameters;

        /// <summary>
        /// Получение интерфейса IRealTimeFrameReceiver.
        /// </summary>
        public GetRTFRDelegate GetRTFR;

        /// <summary>
        /// Получение PTZ интерфейса.
        /// </summary>
        public GetPtzControllerDelegate GetPtzController;

        /// <summary>
        /// Получение I/O интерфейса.
        /// </summary>
        public GetIOControllerDelegate GetIOController;
}
```

Конкретная реализация интерфейса IRealTimeFramReceiver должна возвращаться делегатом *GetRTFR*, вызываемого хостом. В качестве аргументов этому делегату передаются параметры подключения (*ConnectionParameters*) и настройки потоков (*SubStreamParameters*), которые были заданы пользователем в конфигурации канала.
Возможности IP устройства, с которым работает плагин-получатель кадров описываются полем *Capabilities*:

```
/// <summary>
/// Перечень возможностей устройства
/// </summary>
public enum DevType_Capabilities : ulong
{

   /// <summary>
    /// Устройство работает только с камерами.
    /// </summary>
    SupportsCameras = 1,
    /// <summary>
    /// Устройство поддерживает камеры и видеосерверы.
    /// </summary>
    SupportsCamerasAndServers = 2,
    /// <summary>
    /// Устройство поддерживает работу с альтернативным потоком.
    /// </summary>
    SupportsAlternativeVideoStream = 4,
    /// <summary>
    /// Параметры (разрешение, фпс, компрессия), описываемые массивами
    /// SupportedDeviceParameters/SupportedExtraParameters, не зависят от формата
    /// потока (одинаковы для mjpeg, mpeg4, h264).
    /// </summary>
    DeviceParametersFormatIndependent = 8,
}
```

Если требуется решить задачу управления поворотной камерой или ее входами и выходами (IO) необходимо предоставить реализацию соответствующих интерфейсов *IPtzController* и/или *IIOController*:

```
/// <summary>
/// Унифицированный интерфейс реализации управления поворотной камерой.
/// </summary>
public interface IPtzController
{
      #region ----------------- БАЗОВАЯ ЧАСТЬ ------------------

      /// <summary>
      /// Инициализация камеры.
      /// </summary>
      /// <returns></returns>
      void Initialization();

      /// <summary>
      /// Возвращает возможности данной камеры.
      /// </summary>
      /// <returns></returns>
      PtzCapabilities GetCapabilities();

      /// <summary>
      /// Возвращает названия пресетов, установленных на камере.
      /// Количество элементов результирующего массива соответствует количеству
      /// пресетов.
      /// Каждому пресету соответствует номер, равняющийся индексу в массиве.
      /// </summary>
      /// <returns>Названия пресетов, установленных на камере.</returns>
      string[] GetPresetsNames();

      /// <summary>
      /// Устанавливает пресет по его номеру.
      /// </summary>
      /// <param name="presetIndex">Номер пресета.</param>
      void SetPresetPosition(int presetIndex);
```

```
/// <summary>
/// Устанавливает камеру в "домашнее" положение.
/// </summary>
void MoveToHome();

/// <summary>
/// Прекращает выполнение любой команды PTZ.
/// </summary>
void Stop();

/// <summary>
/// Перемещение на шаг.
/// </summary>
/// <param name="panSpeed">Скорость по горизонтали. Интервал от -100 до
/// 100.</param>
/// <param name="tiltSpeed">Скорость по вертикали. Интервал от -100 до
/// 100.</param>
void StepMove(int panSpeed, int tiltSpeed);

/// <summary>
/// Непрерывное(желательно плавное) движение.
/// </summary>
/// <param name="panSpeed">Скорость по горизонтали. Интервал от -100 до
/// 100.</param>
/// <param name="tiltSpeed">Скорость по вертикали. Интервал от -100 до
/// 100.</param>
void ContiniousMove(int panSpeed, int tiltSpeed);

/// <summary>
/// Относительное приближение.
/// </summary>
/// <param name="step">Шаг от 1 до 100.</param>
void StepZoomIn(int step);

/// <summary>
/// Относительное отдаление.
/// </summary>
/// <param name="step">Шаг от 1 до 100.</param>
void StepZoomOut(int step);

/// <summary>
/// Непрерывное (желательно плавное) приближение.
/// </summary>
/// <param name="speed">Скорость. Интервал от 1 до 100.</param>
void ContiniousZoomIn(int speed);

/// Непрерывное (желательно плавное) отдаление.
/// </summary>
/// <param name="speed">Скорость. Интервал от 1 до 100.</param>
void ContiniousZoomOut(int speed);

/// <summary>
/// Возвращает максимальное увеличение камеры.
/// </summary>
/// <returns>Максимальное увеличение камеры. Если функция не
///поддерживается, возвращает отрицательное число.</returns>
double GetMaxZoomFactor();

/// <summary>
/// Возвращает текущее увеличение камеры.
/// </summary>
/// <returns>Текущее увеличение камеры. Если функция не поддерживается,
/// возвращает отрицательное число.</returns>
```

```
        double GetCurrentZoomFactor();

        /// <summary>
        /// Устанавливает абсолютное увеличение. Ничего не делает, если камера это
        /// не поддерживает. См. PtzCapabilities.
        /// </summary>
        void SetZoomFactor(double factor);

        /// <summary>
        /// Устанавливает максимальное увеличение.
        /// </summary>
        void ZoomTele();

        /// <summary>
        /// Устанавливает минимальное увеличение.
        /// </summary>
        void ZoomWide();

        #endregion

        #region ----------------- ДОПОЛНИТЕЛЬНАЯ ЧАСТЬ ------------------

        /// <summary>
        /// Поворачивает камеру таким образом, что указанная точка оказывается в
        /// центре области кадра.
        /// </summary>
        /// <param name="point">Точка изображения, которую необходимо поместить в
        /// центр путём поворота камеры. Задаётся в пикселях.
        /// Начало отсчета(0,0) - левый верхний угол кадра о</param>
        /// <param name="frameSize">Размер кадра в пикселях</param>
        void MoveTo(System.Drawing.Point point, System.Drawing.Size frameSize);

        /// <summary>
        /// Поворачивает и масштабирует камеру таким образом,
        /// что указанный прямоугольник занимает всю область кадра.
        /// Если пропорции прямоугольника не соответсвуют пропорциям кадра, то
        /// масштабирование производится таким образом, чтобы прямоугольник весь
        ///вошёл в кадр.
        /// Центр прямоугольника помещается в центр кадра.
        /// </summary>
        /// <param name="rect">Прямоугольник, задается в пикселях</param>
        /// <param name="frameSize">Размер кадра в пикселях</param>
        void ShowRect(System.Drawing.Rectangle rect, System.Drawing.Size frameSize);

        #endregion
}

/// <summary>
/// Уинифицированный интерфейс реализации управления входами/выходами камеры
/// </summary>
public interface IIOController
{
    /// <summary>
    /// Инициализация
    /// </summary>
    /// <returns></returns>
    void Initialization();

   /// <summary>
   /// Возвращает возможности данной камеры.
   /// </summary>
   /// <returns></returns>
   IOCapabilities GetCapabilities();
```

```
  /// <summary>
  /// Устанавливает заданное значение на выходе
  /// </summary>
  /// <param name="portID">Номер выхода</param>
  /// <param name="value">1 или 0</param>
  void SetOutput(int portID, int value);

  /// <summary>
  /// Выдает последовательность импульсов (ШИМ) на указанном выходе.
  /// Поддерживается не всеми камерами.
  /// </summary>
  /// <param name="portID">Номер выхода</param>
  /// <param name="pulses">Массив импульсов</param>
  void SetOutput(int portID, IOPulse[] pulses)
}
```

Возможности поворотного устройства (IO контроллера) должны возвращаться методом *GetCapabilities()* для информирования хоста о том, какие методы опциональных возможностей (если поддерживаются устройством) реализованы в интерфейсе.

Регистрационную информацию *DevType_RegInfo* устройства необходимо указывать на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс IPlugin, вызвать метод *RegisterDevType* интерфейса хоста (см. также 0).

Помимо *DevType_RegInfo*, необходимо также заполнить информацию о получателе кадров *RTFR_RegInfo*:

```
/// <summary>
/// Регистрационная информация о получателе кадров
/// </summary>
public class RTFR_RegInfo
{
	/// <summary>
	/// Идентификатор устройства
	/// </summary>
	public Guid DeviceTypeGuid;

	/// <summary>
	/// Формат потока данных
	/// </summary>
	public VideoStreamFormats StreamFormat;

	/// <summary>
	/// Используемый протокол подключения
	/// </summary>
	public NetworkConnectionTypes ConnectionType;

	/// <summary>
	/// Возможности устройства при заданном формате StreamFormat
	/// </summary>
	public RTFR_Capabilities Capabilities;
}
```

Данную информацию нужно задать столько раз, сколько различных форматов потока поддерживает IP-устройство. Аналогично *DevType_RegInfo*, информацию *RTFR_RegInfo* необходимо задавать на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс IPlugin. Для этого достаточно вызвать метод *RegisterDevType* интерфейса хоста. Пример заполнения данных классов приведен ниже:

```
public void Initialize(IPluginHost host) {
```

```
DevType_RegInfo KingNet_KS3002MJ_Device = new DevType_RegInfo();
KingNet_KS3002MJ_Device.DeviceTypeGuid =
                                    new Guid("5C49C51D-B17B-40b0-9656-6DC71DD86D90");
KingNet_KS3002MJ_Device.DevTypeModelName = "KS3002MJ";
KingNet_KS3002MJ_Device.DevTypeBrandName = "KingNet";
KingNet_KS3002MJ_Device.AvailableResolutions = GetConcreteResolutions();
KingNet_KS3002MJ_Device.Capabilities = DevType_Capabilities.SupportsCameras;
KingNet_KS3002MJ_Device.GetPtzController = null;
KingNet_KS3002MJ_Device.GetRTFR = (conParam, subParams) =>
{
     RealTimeFrameReceiver rtfr = new RealTimeFrameReceiver(conParam, subParams);
     return rtfr;
};
KingNet_KS3002MJ_Device.SetDeviceParameters = (conParams, subParams) =>
{
     return true;
};

host.RegisterDevType(KingNet_KS3002MJ_Device);

RTFR_RegInfo rtfrKingNet_KS3002MJ_Mjpeg = new RTFR_RegInfo();
rtfrKingNet_KS3002MJ_Mjpeg.DeviceTypeGuid =
                                    new Guid("5C49C51D-B17B-40b0-9656-6DC71DD86D90");
rtfrKingNet_KS3002MJ_Mjpeg.StreamFormat = VideoStreamFormats.MJPEG;
rtfrKingNet_KS3002MJ_Mjpeg.ConnectionType = NetworkConnectionTypes.UDP;
rtfrKingNet_KS3002MJ_Mjpeg.Capabilities.SupportedStreamTypes =
                                                    ChannelStreamTypes.MainVideo;
rtfrKingNet_KS3002MJ_Mjpeg.Capabilities.SupportedExtraParameters = new DeviceParameters[]
                                    { DeviceParameters.Null, DeviceParameters.Null };
rtfrKingNet_KS3002MJ_Mjpeg.Capabilities.SupportedDeviceParameters = new
                    DeviceParameters[] { DeviceParameters.Null, DeviceParameters.Null };

host.RegisterRTFR(rtfrKingNet_KS3002MJ_Mjpeg);
}
```

5 Интерфейсы HTTP и RTSP

5.1 HTTP-интерфейс

Наиболее простым способом получения потоков данных из **MACROSCOP** являются HTTP интерфейс. Получение видео реального времени по HTTP осуществляется следующим общего вида CGI-запросом на сервер:

http://<ip адрес и порт сервера macrosop>/video?channel="<название канала>"[&channelid=id]&login="<имя пользователя>"&password="хэш-строка MD5 пароля"[&sound=on][&streamtype=alternative]

Если у пользователя нет пароля, то параметр password можно опустить или оставить его значение пустым. Опциональный параметр sound со значением on позволяет вместе с видео в том же соединении получать звук путем чередования кадров. Звуковые кадры всегда приходят в формате G.711U. Опционально можно запрашивать альтернативный поток, который, как правило, идет в меньшем разрешении и может быть использован для отображения.

В результате на указанный выше запрос сервер шлет "бесконечный" HTTP ответ, в котором идут видео (и аудио) кадры, разделенные заголовками. Типичный ответ сервера на запрос:

HTTP/1.1 200 OK

…

Content-Type: multipart/x-mixed-replace; boundary=myboundary

-- myboundary
Content-Type: image/jpeg
Content-Length: 63125

<тело JPEG кадра>

или

-- myboundary
Content-Type: audio, PCMU
Content-Length: 1000

<тело G711U кадра>

Если на канале установлен формат потока MJPEG, то в значении параметра Content-Type содержится строка «image/jpeg». В случае MPEG4 передается Content-Type равный «video, mpeg4, I-frame» в случае I-кадров и «video, mpeg4, P-frame» для P-кадров. В каждом опорном I-кадре имеется инициализирующая информация для декодера MPEG4. Аналогично, в случае H.264, для I-кадров в поле Content-Type содержится значение «video, h264, I-frame» и «video, h264, P-frame» для P-кадров. Как и в случае MPEG4, перед каждым I-кадром встраивается инициализирующая информация для декодера H264.

Пример запроса для получения видео реального времени: http://127.0.0.1:8080/video?channel="Канал 1"&login=root&password=""

Во избежание коллизий, рекомендуется вместо имени канала передавать его идентификатор в параметре channelid:

http://127.0.0.1:8080/video?channelid=7f54efa4-e7d6-456e-ac10-18215f2492d6&login=root&password=""

Для того чтобы узнать идентификаторы каналов и их настройки в **MACROSCOP**, необходимо выполнить запрос:

http://<ip адрес и порт сервера macrosop>/configex?login="<имя пользователя>"&password="хэш-строка MD5 пароля"

Пример запроса:

http://127.0.0.1:8080/configex?login=root&password=

Ответ на запрос приходит в xml-формате:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://www.macroscop.com" XMLProtocolVersion="2" Timestamp="2012-07-16T02:28:03.523637Z" Revision="13" Id="e455524d-81ba-4efc-a6ca-bbc91d859f66" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" SenderId="17a73597-580f-41d2-81ec-35ab7d78fd6a">
<Servers>
    <ServerInfo Id="17a73597-580f-41d2-81ec-35ab7d78fd6a" Name="Новый сервер 1" Url="127.0.0.1:8080"/>
</Servers>
<Channels>
<ChannelInfo Id="d55e63b2-0061-4a2c-aba5-66b7ee97e1eb" DeviceInfo="Axis Mxxxx, Pxxxx, Qxxxx" IsSoundArchivingEnabled="true" IsArchivingEnabled="true" IsSoundOn="false" IsDisabled="false" Name="Канал 1" AttachedToServer="17a73597-580f-41d2-81ec-35ab7d78fd6a">
    <Streams>
        <StreamInfo StreamFormat="H264" StreamType="Main"/>
        <StreamInfo StreamFormat="MJPEG" StreamType="Alternative"/>
    </Streams>
</ChannelInfo>
</Channels>
</Configuration>
```

В xml-ответе в элементе *Configuration* содержится:

1) Версия протокола *XMLProtocolVersion*. На данный момент номер протокола равняется 2, его смена в будущем предполагает появления новых элементов и атрибутов в xml-ответе.
2) Временная метка последнего применения конфигурации *Timestamp*
3) Уникальный идентификатор *Id* текущей конфигурации и номер ее ревизии *Revision*. Номер ревизии увеличивается на единицу после каждого изменения конфигурации.
4) Идентификатор сервера *SenderId*, отправившего данный xml-ответ

В элементе *Servers* содержится описание серверов, входящих в текущую конфигурацию. Каждый сервер описывается элементом *ServerInfo*, в который входят следующие атрибуты:

1) Уникальный идентификатор сервера *Id*
2) Его название *Name* в рамках текущей конфигурации
3) *Url* сервера

В элементе *Channels* содержится описание настроек каналов текущей конфигурации. Настройки каждого канала описываются элементом *ChannelInfo*, в котором содержатся следующие атрибуты и элементы:

1) Уникальный идентификатор *Id* канала, который может быть использован в запросах на получение видео при задании параметра *channelid*
2) Идентификатор сервера *AttachedToServer*, к которому прикреплен канал
3) Информация о выбранном устройстве *DeviceInfo*
4) Параметр *IsArchivingEnabled*, показывающий включено ли на канале архивирование видеоданных
5) Параметр *IsSoundArchivingEnabled*, показывающий включено ли архивирование звуковых данных.
6) Параметр *IsSoundOn*, показывающий включено ли получение звука на данном канале. Если данный параметр равен *false*, то параметр *IsSoundArchivingEnabled* может быть проигнорирован
7) Параметр *IsDisabled*, показывающий отключен ли канал в текущей конфигурации
8) Параметр *Name*, содержащий имя канала в конфигурации
9) Элемент *Streams*, содержащий настройки основного и альтернативного потоков видеоданных. В описании потока *StreamInfo* содержится один из трех возможных форматов потоков H.264/MPEG4/MJPEG в поле *StreamFormat* и тип потока, принимающий значение *main (основной)* или *alternative (альтернативный)*. Если альтернативный поток не включен на данном канале, то его описание будет отсутствовать.

Для доступа в архив по HTTP интерфейсу достаточно сформировать следующий CGI-запрос на сервер:

http://<ip адрес и порт сервера macrosop>/mjpeg?mode=archive&startTime=”dd:mm:yyyy hh:mm:ss”[&speed=n]&channel=”<название канала>”[&channelid=id]&login=”<имя пользователя>”&password=”хэш-строка MD5 пароля”[&sound=on]

Параметр startTime является стартовой позицией, с которой начинается воспроизведение архива. Его значение представляется в виде комбинации даты и местного (локального) времени.

Опциональный параметр speed задает скорость воспроизведения архива. Диапазон принимаемых значений является непрерывным и изменяется от 0.1 до 20. Значение по умолчанию 1.0.

Пример запроса: http://127.0.0.1:8080/mjpeg?mode=archive&startTime=”20.09.2011 11:49:12”&speed=1&channel="Канал 1"&login=root&password=""

5.2 RTSP-интерфейс

RTSP-интерфейс используется для передачи видео (и звука) клиентам, работающих по RTSP-протоколу. Данный интерфейс поддерживает формат .H264. По умолчанию

интерфейс отключен в **MACROSCOP**, для его активации требуется запустить конфигуратор и в разделе серверных настроек в подразделе «Сетевые настройки сервера» отметить флажок «Принимать подключения по протоколу RTSP (для вещания .H264)».

Подключение к серверу осуществляется rtsp клиентом, например VLC, с помощью следующей строки подключения:

rtsp://<ip адрес сервера и порт rtsp>/channel=<название канала>[&channelid=id][&sound=on]

Кавычки в строке подключения не допускаются. В процессе подключения к серверу выполняется Digest-аутентификация средствами RTSP.

Пример запроса:

rtsp://127.0.0.1/channel=Канал%201

Допускается заменять символ пробела на %20. Для доступа в архив требуется задать параметры аналогично параметрам при подключении по HTTP.

Пример запроса:

rtsp://127.0.0.1/mode=archive&channel=Канал%201&startTime=20.09.2011%2011:49:12&speed=1

6 Интерфейс XML

XML интерфейс позволяет посылать на сервер **MACROSCOP** запросы в формате XML и получать в ответ данные в том же формате.

Структура запроса должна быть следующей:

```
<?xml version="1.0" encoding="utf-8" ?>
<query>
  <server_login>root</server_login>
  <server_pass_hash></server_pass_hash>
  <query_name>get_people_counters</query_name>
  <query_params>
    …
  </query_params>
</query>
```

Ниже описано назначение параметров:

Параметр	Описание
server_login	Имя пользователя, под которым будет выполняться команда.
server_pass_hash	md5 хэш пароля пользователя.
query_name	Строковое наименование типа запроса.
query_params	Внутри данного тэга будут помещаться параметры, специфичные для типа запроса, указанного в параметре query_name.

В ответ сервер возвращает ответ вида:

```
<?xml version="1.0" encoding="utf-8" ?>
<result>
  <query_name></query_name>
  <query_result>Ok</query_result>
  <query_msg>Запрос выполнен успешно.</query_msg>
  <query_time>20.09.2012 10:58:15</query_time>
  <query_time_local>20.09.2012 16:58:15</query_time_local>
</result>
```

Ниже описано назначение параметров:

Параметр	Описание
query_name	Строковое наименование типа запроса.
query_result	«Ok» - если запрос выполнен успешно, «Error» - если произошли какие-либо ошибки.
query_msg	Строковый комментарий по результатам выполнения запроса.
query_time	Время запроса UTC.
query_time_local	Время запроса локальное.

В ответе также могут быть тэги, специфичные для конкретного типа запроса.

6.1 Получение данных счётчика посетителей

Для получения данных счётчика посетителей используется запрос *get_people_counters*.

Параметрами данного запроса являются:

```
<channel_id>cacdd8e6-1c56-435c-86e3-6967d7494a50</channel_id>
<search_time>2012-09-17 09:50:00</search_time>
```

Параметр	Описание
channel_id	Идентификатор канала, на котором настроен счётчик посетителей.
search_time	Момент времени, для которого необходимо выдать показания счётчика. Время указывается в формате yyyy-MM-dd HH:mm:ss. Время должно указываться по гринвичу (UTC).

В ответ сервер возвращает следующие параметры:

```
<in>434</in>
<out>378</out>
```

Параметр	Описание
in	Количество вошедших людей для данного счётчика.
out	Количество вышедших людей для данного счётчика

7 Организация вещания видео на сайт

Вещание видео на сайт может быть организовано с помощью мобильного сервера на стороне MACROSCOP и JavaScript на клиентской стороне. Скрипт для клиентской стороны и пример его использования в html-странице может быть найден по пути Examples\Site\frameReceiver.js. В скрипте необходимо установить параметры подключения к серверу MACROSCOP, номер или идентификатор канала с которого должно транслироваться видео и желаемый размер области, в которую будут выводиться видеокадры.

Пример настройки скрипта:

```
var serverUrl = "http://95.23.84.1:8080" /*URL сервера*/
var login = "root" /*пользователь, имеющий права на просмотр транслируемого канала*/
var password = ""; /*MD5-хэш пароля пользователя в верхнем регистре или пуста строка, если пароль пустой*/
var channelnum = 0; /*порядковый номер канала в общей конфигурации, счет с 0*/
var drawWidth = 577; /*ширина области отображения, в пикселях*/
var drawHeight = 432; /*высота области отображения, в пикселях*/
```

Вместо передачи порядкового номера канала, имеется возможность задать идентификатор канала. Идентификаторы всех каналов в системе могут быть получены с помощью соответствующего запроса (см. главу 5.1). Идентификатор канала передается с помощью с параметра channelid в запросе. Пример скрипта, использующего идентификатор канала вместо его порядкового номера, находится по пути Examples\Site\frameReceiver_id.js. На самой html-странице в нужном месте должен быть тэг , в этом месте будет отображаться MJPEG видеопоток.

Замечания:

1) Не рекомендуется изменять размеры области отображения видео динамически. Это приведет к существенному повышению потребляемых ресурсов со стороны мобильного сервера MACROSCOP. Делом в том, что мобильный сервер перекодирует видеопоток с канала в MJPEG и вновь полученный поток кадров разделяет между многими клиентами (браузерами). Использование разных разрешений (размеров областей отображения) приведет к дополнительной загрузке мобильного сервера MACROSCOP.
2) По умолчанию сервер MACROSCOP является промежуточным звеном между клиентом (браузером) и мобильным сервером. В целях снижения загрузки основного сервера, имеется возможность включить перенаправление запросов клиентов напрямую на мобильный сервер. Для этого в конфигураторе в серверных настройках на вкладке «Мобильные устройства» снять флажок «Подключение к серверу мобильных устройств в режиме прокси». Однако в этом случае потребуется сделать доступным порт мобильного сервера для внешних клиентов.