Macroscop SDK

Опубликовано: 19.05.2026

Оглавление

Введение

В предыдущих версиях данный документ содержал часть запросов **Macroscop HTTP API**. Теперь эти запросы перенесены в документ **Macroscop REST API**, скачать который можно с сайта doc.macroscop.com на странице Документация для скачивания.

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 можно обратиться к инструкциям оператора и администратора, поставляемых в комплекте с **Macroscop**.

С версии Macroscop 4.0 используется версия платформы Net6.
Плагины для версий 3.6 и ранее должны быть пересобраны под новую версию платформы.

В **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:

Имя плагина	Название	Описание
ExternalAction	Действие	Базовый класс, позволяющий добавлять новые действия для сценариев и планировщика задач по расписанию
VideoAnalyst	Видеоаналитика	Базовый класс для осуществления видеоаналитики на сервере
PluginVisualiser	Визуализатор	Базовый класс визуализатора для графического отображения специфической информации на канале в приложении Macroscop Клиент
IClientPluginMenuItem	Элемент меню	Интерфейс, позволяющий создавать в приложении **Macroscop Клиент** собственный подпункт в меню **Настройка**
EventProcessor	Процессор событий	Базовый класс процессора событий. Позволяет регистрировать и генерировать собственные события, получать события из **Macroscop**, а также выполнять команды в канале. Плагины данного типа могут быть использованы для осуществления интеграции с другими системами.
ICameraServiceProvider	Получатель кадров	Интерфейс получателя кадров с IP-устройств. Позволяет получать с IP-устройств (камер) видео, звук, данные детекции движения; а также управлять поворотными камерами.

Указанные типы базовых классов (интерфейсов), а также некоторые другие вспомогательные сущности подробно рассматриваются 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 программного комплекса (см. Регистрация плагинов в Macroscop).

Типовые задачи

1. **Интеграция IP-камер**

   Для подключения IP-устройства (камеры) достаточно реализовать плагин **Получатель кадров**. Сведения о данном типе плагина предоставлены в разделе Плагин-получатель кадров. Шаблон плагина размещен в папке с примерами, в проекте **Camera.csproj**.

2. **Интеграция со СКУД, ОПС, POS-терминалами и т.п.**

   Интеграция может быть выполнена через плагин **Процессор событий**, способный получать события от **Macroscop**, генерировать в **Macroscop** собственные события в процессе взаимодействия с другой системой, выполнять в **Macroscop** команды в канале (включение/выключение записи, установка пресетов, ввод-вывод с камер и др.), получать доступ к архиву **Macroscop**. Сведения о данном типе плагина предоставлены в разделе Плагин Процессор событий. Пример плагина размещен в папке с примерами, в проекте **EP_PluginExample.csproj**. Если требуется получать только видео и звук из **Macroscop**, то можно использовать более простой вариант, рассмотренный в разделе HTTP-интерфейс для получения видео; пример получения видео по HTTP размещен в папке с примерами, в проекте **HttpVideo.csproj**.

3. **Видеоаналитика**

   Любой алгоритм обработки видеопотока может быть реализован с помощью плагина **Видеоаналитика**. Все результаты работы данного плагина представляются событиями, которые могут быть далее интерпретированы плагинами **Визуализатор** и **Элемент меню**. Указанные плагины описаны в разделах Плагин Видеоаналитика, Плагин Визуализатор и Плагин Элемент меню; пример использования размещен в папке с примерами, в проекте **Analyst_PluginExample.csproj**.

Краткое описание проектов с примерами использования Macroscop SDK

Каждый проект имеет отдельное описание работы в файле README.md.

Также описание и примеры реализации некоторых плагинов находятся в нашем [репозитории на GitHub](#).

Название проекта	Описание
Analyst_PluginExample	Плагин добавляет интеллектуальный модуль **Test Event Module**, который генерирует событие **CounterEvent**, отображаемое в журнале событий **Macroscop Клиента**.
Camera	Плагин добавляет возможность получать тестовое видео с тестовой камеры: **Macroscop Конфигуратор** -> Камеры -> Добавить камеру -> Производитель = **Sample Brand**, Тип устройства = Камера, Модель = **Sample Camera**. В качестве примера, на видео постоянно отображается текст **SAMPLE FRAME**.
EP_PluginExample	Плагин добавляет модуль **External events generating module**, который генерирует событие **CounterEvent**, отображаемое в журнале событий **Macroscop Клиента**. В событии содержится значение счетчика, которое инкрементируется в соответствии с заданной в настройках частотой.
ExternalAction_PluginExample	Плагин добавляет действие **Test external action** по событию в сценариях и планировщике задач по расписанию. Действие записывает сообщение в лог **Test external action module.log**.
HttpInterface	В примере реализовано отображение списка каналов сервера, списка зарегистрированных событий. Также доступно получение событий в реальном времени и специальных архивных событий.
HttpVideo	Пример демонстрирует получение видео **(mjpeg)** с камеры с помощью **HTTP**-интерфейса.
MenuItem_PluginExample	Плагин добавляет элемент меню в **Клиенте** (Левая панель -> Дополнительно -> MENU ITEM EXAMPLE). При нажатии на элемент меню происходит открытие окна, в котором содержатся идентификаторы каналов.
RtVisualizer	Плагин отображает информацию в ячейке наблюдения камеры в **Клиенте**, при возникновении события тревоги на камере.

Плагины

В данной главе описан процесс создания и регистрации плагинов, а также подробно рассмотрен каждый тип плагина. Наиболее важные фрагменты кода отражены в тексте.

Пример создания простого плагина

Регистрация плагинов в Macroscop

В момент запуска отдельных приложений — компонентов **Macroscop** (**Сервер**/**Клиент**/**Конфигуратор**; далее в тексте — **хост**), — происходит поиск .NET сборок в папке **Plugins** запускаемого приложения. 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 : IService
{
        /// <summary>
        /// Получает интерфейс менеджера протоколов
        /// </summary>
        /// <returns></returns>
        IMcLogMgr GetLogManager();

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

// ... Прочие, не показанные здесь, функции регистрации
}
```

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.

Сериализуемые классы в плагинах

Плагин может содержать сериализуемые классы, например наследники `IAction`, `RawEvent`, классы настроек. Такие классы обычно помечены аттрибутом `[Serializable]`. В плагинах такие классы дополнительно необходимо помечать аттрибутом `[AlarusSerializable]`.

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

      /// <summary>
      /// Описание на основе текущих настроек
      /// </summary>
      public abstract string Description
      {
            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(RawEvent rawEvent);

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

      /// <summary>
      /// Позволяет генерировать событие, заполняется хостом перед вызовом действия
      /// </summary>
```

```
        [NonSerialized]
        public GenerateEventDelegate GenerateEvent;
    }
```

В классе наследнике требуется задать следующие атрибуты:

- **ActionGUIName** — название действия, которое будет фигурировать в графическом интерфейсе в конфигураторе.
- **GuidAttribute** — идентификатор действия.

Опционально 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);

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

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**). Подробная информация о доступных командах приведена в разделе Плагин Процессор событий 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.

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 **Действие** в системе, необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс **IPlugin**, вызвать метод **RegisterExternalAction** интерфейса хоста (см. Регистрация плагинов в Macroscop).

Для действий, имеющих настройки в графическом интерфейсе, необходимо добавить атрибут **ActionAssemblyPluginHasSettingsControl** в файл **AssemblyInfo.cs.**

Плагин Видеоаналитика

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="id">Идентификатор канала</param>
  /// <param name="analystToolSet">Тул сет аналитики</param>
  /// <param name="pluginEnv">Настройки плагина</param>
  public abstract void Initialize(Guid id, IPluginAnalystToolSet analystToolSet,
PluginEnvironment pluginEnv);

  /// <summary>
  /// Метод обработки кадра и карты движения. Вызывается хостом.
  /// </summary>
  /// <param name="image">Кадр</param>
  /// <param name="motionMap">Карта движения, может быть null</param>
  /// <param name="background">Фон от детектора движения. Равен null, если
  ///NeedBackground == false</param>
  public abstract void Process(ImageData image, MotionMap motionMap,
BackgroundImage background);

  /// <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 => false;

  public virtual bool NeedBackground => false;

  public virtual TimeSpan MinTimeBetweenFrames => TimeSpan.Zero;

  public virtual TimeSpan MaxTimeBetweenFrames => TimeSpan.MaxValue;

  protected ServerRole ServerRoleForChannel { get; set; }

  /// <summary>
  /// Поддерживаемый формат пикселя
  /// </summary>
  public virtual VAPixelFormat PixelFormat => VAPixelFormat.BGR24;

  /// <summary>
  /// Выполняет команду.
  /// </summary>
  /// <param name="cmdObj"></param>
  /// <returns></returns>
  public virtual object ProcessCommand(object cmdObj)
  {
     return null;
  }

  public SendServerCommandDelegate SendServerCommand;

  /// <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="channel"></param>
  /// <param name="settingsHost"></param>
  /// <param name="settings">Текущие параметры аналитика</param>
  /// <returns>Новые параметры аналитика</returns>
  public abstract PluginSettings SetSettings(Guid channel, ISettingsHost
settingsHost, PluginSettings settings);

  public virtual object GetAdditionalInfo()
  {
     return null;
  }

  public virtual object GetAdditionalInfo(params object[] parameters)
  {
     return null;
  }

  /// <summary>
  /// Возвращает конфигурацию в виде массива байт
  /// </summary>
  /// <returns></returns>
  public virtual byte[] GetBinarySettings()
  {
     return null;
```

```
  }

  /// <summary>
  /// Изменяет общие или специфические настройки аналитика, если это необходимо.
  /// </summary>
  /// <param name="binarySettings">настройки аналитика в виде массива байт</param>
  public virtual void SetSettings(byte[] binarySettings)
  {
  }

  public virtual List<VideoAnalystZone> GetZones(object channelSpecificSettings)
  {
     return new List<VideoAnalystZone>();
  }

  public virtual bool NeedReferencedImage()
  {
     return false;
  }

  /// <summary>
  /// Нужны ли вообще аналитику видеокадры? (Некоторые аналитики, например,
  /// 3D-подсчет сейчас получают с камеры готовые данные. В этом случае видео
тянуть
  /// не надо).
  /// </summary>
  /// <returns></returns>
  public virtual bool NeedVideoFrames()
  {
     return true;
  }

  /// <summary>
  /// Конвертирует настройки аналитика из Json <paramref name="settingsJson"/> с
  /// учетом текущих настроек из
  /// конфигурации <paramref name="currentSettings"/> и возвращает новые настройки
  /// аналитика.
  /// </summary>
  /// <param name="settingsJson">Новые настройки аналитика в формате Json</param>
  /// <param name="currentSettings">Текущие настройки аналитика</param>
  /// <param name="converter">Json конвертер</param>
  /// <returns>Новые настройки аналитика</returns>
  public virtual PluginSettings ConvertJsonSettingsToPluginSettings(string
settingsJson,
     PluginSettings currentSettings, IJsonConverter converter) =>
default(PluginSettings);

  /// <summary>
  /// Конвертирует текущие настройки аналитика <paramref name="currentSettings"/> в
  /// Json и возвращает их.
  /// </summary>
  /// <param name="currentSettings">Текущие настройки аналитика</param>
  /// <param name="converter">Json конвертер</param>
  /// <returns>Настройки в формате Json</returns>
  public virtual string ConvertToJsonSettings(PluginSettings currentSettings,
IJsonConverter converter) => null;

}
```

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 аналитики, отображаемой в графической оболочке приложения **Macroscop Конфигуратор**. Если аналитика может быть настроена в конфигураторе, необходимо реализовать метод настройки **SetSettings**, и указать атрибут **PluginHasSettingsAttribute**.

Данные изображений подаются на вход аналитики в виде класса **ImageData.**

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 аналитики едины для всех каналов, то достаточно заполнять только объект общих настроек. В противоположном случае, когда все настройки аналитики 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 аналитики хранятся в общей конфигурации **Macroscop**.

Рассмотрим также метод **ProcessCommand**, который позволяет аналитике выполнять команды от плагина **Элемент меню** (см. Плагин Элемент меню). 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 **Видеоаналитика** работает на сервере, а плагин **Элемент меню** — всегда в клиенте.

В процессе обработки видеокадров аналитика должна передавать результаты своего анализа серверу с помощью генерации событий. Для этого необходимо заранее зарегистрировать на стороне хоста (см. Регистрация плагинов в Macroscop) 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 **RawEvent**:

```
/// <summary>
/// Родительский класс в иерархии событий, происходящих в канале.
/// </summary>
[Serializable][AlarusSerializable]
public abstract class RawEvent : IDescribable
{
```

```
    /// <summary>
    /// ВременнАя метка события.
    /// </summary>
    [OptionalField]
    public DateTime EventTime;

    /// <summary>
    /// Комментарий к событию.
    /// </summary>
    [OptionalField]
    public string Comment;

    public virtual EventTextDescription GetDescription(IToStringConverter converter)
    {
        return new EventTextDescription("");
    }

    public virtual object GetAttachments()
    {
        return null;
    }

    /// <summary>
    /// Создание сырого события
    /// </summary>
    protected RawEvent()
    {
        EventTime = DateTime.UtcNow;
    }
}
```

Каждое пользовательское событие должно иметь ряд обязательных атрибутов:

- **GuidAttribute** — определяет в явном виде уникальный идентификатор события;
- **Serializable** и **AlarusSerializable** — позволяют выполнять сериализацию события.

Кроме того, имеется ряд необязательных (опциональных) атрибутов:

- **EventLocalizedName** — задает название события в графическом интерфейсе хоста. Если атрибут не задан, то событие не будет отображаться в конфигураторе в разделе сценариев;
- **EventSaveable** — задает 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**, **UnifiedLog** или **SpecialAndUnifiedLog**). Свойство события **GenerationFrequency** задает частоту генерации данного события, требует параметр **EventGenerationFrequency** (см. ниже). Данное свойство влияет на работу сервера при сохранении событий в БД и имеет рекомендательный характер. Если свойство не было указано, то по умолчанию используется режим **Middle**. Этот режим является предпочтительным. Режим **Low** не рекомендуется для использования, поскольку данные события записываются в особую, критичную для функционирования, базу данных;
- **EventGeneratesAlarmByDefault** — событие по умолчанию является тревожным, что означает автоматическую привязку к событию действия **Генерация тревоги** при создании нового канала в конфигураторе **Macroscop**.

Если пользовательское событие имеет поля, которые должны сохраняться в БД, необходимо для каждого из полей указывать атрибут **EventFieldSaveableOrScenariesUsable**. В качестве параметров атрибут требует указать **orderNum** (порядковый номер поля, отсчет начинается с 0) и **indexable** (флаг, 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 **RawEvent**, 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**.

Пользовательское событие может быть представлено следующим образом:

```
[GuidAttribute("389EDCE2-54BB-4C2C-9984-51B7516A5DDF")]
[EventLocalizedName("Оставлен предмет")]
[EventSaveable(EventSaveMode.SpecialAndUnifiedLog, "objectleft")]
[EventGeneratesAlarmByDefault]
[Serializable][AlarusSerializable]
public class ObjectLeftEvent : RawChannelEvent
{
   [EventFieldSaveableOrScenariesUsable(0, true)]
   [EventFieldLocalizedName("Предмет")]
   private string objectName;

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

   public override EventTextDescription GetDescription(IToStringConverter converter)
   {
       var eventText = "Оставлен предмет";
        var eventDescription = new EventTextDescription(eventText);

      return eventDescription;
   }

}
```

Для регистрации плагина **Видеоаналитика** необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс **IPlugin**, вызвать метод **RegisterAnalyst** интерфейса хоста (см. Регистрация плагинов в Macroscop).

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 приложения **Macroscop Клиент**. Для создания данного типа плагина необходимо создать наследника класса **PluginVisualiser**:

```
/// <summary>
/// Класс визуализатора.
/// </summary>
public abstract class PluginVisualiser
{
  public abstract DrawingVisual NullableDrawingVisual { get; }

  /// <summary>
```

```
  /// Панель для отрисовки примитивов, текста и другой информации на канале.
  /// </summary>
  protected IDrawingPanel DrawPanel;

  /// <summary>
  /// Контейнер графических элементов. Позволяет размещать отдельные
  /// UserControl'ы в канале.
  /// </summary>
  protected Canvas ControlsContrainer;

  protected IPluginClientToolSet PluginClientToolset;

  public readonly Guid AnalystPluginId;

  public bool ArchiveMode;

  public virtual IVideoTransformer VideoTransformer
  {
     get { return null; }
  }

  protected PluginVisualiser(Guid analystPluginId)
  {
     AnalystPluginId = analystPluginId;
     ArchiveMode = false;
  }

  /// <summary>
  /// заданы ли пользователем настройки визуализатора
  /// </summary>
  protected bool _hasUserSettings;

  public bool HasUserSettings
  {
     get { return _hasUserSettings; }
  }

  /// <summary>
  /// Инициализация визуализатора. Вызывается хостом.
  /// </summary>
  public virtual void Initialize(Guid channelId, IPluginClientToolSet
pluginClientToolset, IPluginVisualizerSet visualiserSet)
  {
     PluginClientToolset = pluginClientToolset;
     DrawPanel = visualiserSet.DrawingPanel;
     ControlsContrainer = visualiserSet.ControlsContrainer;
     RegisterPluginMenuAction = visualiserSet.AddMenuActionHandler;

     if (visualiserSet.VisualizeParams.ParamsRef == null)
        _hasUserSettings = false;
     else
        _hasUserSettings = true;
  }

  /// <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, RawEvent chEv, bool isAlarm);
```

```
  /// <summary>
  /// Делегат для регистрации подпункта во всплывающем меню канала в клиенте
  /// Macroscop.
  /// Заполняется хостом. Вызывается визуализатором.
  /// </summary>
  protected RegisterPluginMenuActionHandler RegisterPluginMenuAction;

  /// <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()
  {
     // сделал, это чтобы очищались ресурсы визуализаторов, т.к. по факту
реализации
     // не перегружают Release,а копипастить неохото.
     Clear();

     DrawPanel = null;
     ControlsContrainer = null;
     PluginClientToolset = null;
  }

  public void SetPresentationModeIfNeed()
  {
     if (!HasUserSettings)
        SetPresentationMode();
  }

  /// <summary>
  /// Переопределить в визуализаторе плагина, если требуется автоматическое
  /// отображение результатов/настроек плагина в демо-версии
  /// </summary>
  protected virtual void SetPresentationMode()
  {

  }

  public virtual void SetDigitalZoomStatus(bool isDigitalZoomActive)
  {
  }
}
```

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 приложении **Macroscop Клиент**. Это позволяет изменять логику работы визуализатора в зависимости от предпочтений пользователя. Данная возможность предоставляется делегатом **RegisterPluginMenuAction**.

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

Плагин Элемент меню

Плагин данного типа позволяет создавать собственный графический интерфейс, открываемый через подпункт меню **Дополнительно** приложения **Macroscop Клиент**.

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 реализовать интерфейс **IClientPluginMenuItem**:

```
public interface IClientPluginMenuItem
{
  void ShowWindow(IPluginClientToolSet pluginToolSet);

  string Header { get; }

  string ImageName { get; }

  bool HasActivePlugins(IPluginClientToolSet pluginToolSet);
}
```

В классе-наследнике необходимо определить метод **ShowWindow**, в который со стороны хоста (приложения **Macroscop Клиент**) 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 IPluginClientToolSet
{
  /// <summary>
  /// Получает информацию о том, доступно ли текущему пользователю редактирование
  /// данных в клиенте в интеллектуальных плагинах
  /// </summary>
  bool CanAccessEditingAnalystPluginsInClient { get; }

  /// <summary>
  /// Получает информацию о том, доступно ли текущему пользователю доступ к базам
  /// данных
  /// </summary>
  bool CanUseDatabase { get; }

  /// <summary>
```

```
  /// Получает информацию о том, доступны ли текущему пользователю отчеты
  /// </summary>
  bool CanUseReports { get; }

  /// <summary>
  /// Получает интерфейс для чтения конфигурации
  /// </summary>
  /// <returns></returns>
  IPluginsConfigReader ConfigReader { get; }

  /// <summary>
  /// Получает интерфейс для работы с архивом
  /// </summary>
  /// <returns></returns>
  IArchiveEventsReader ArchiveEventsReader { get; }

  /// <summary>
  /// Предоставляет перевод идентификаторов сущностей(каналы, пользователи, модули
  /// аналитики, события)
  /// в строковое представления для GUI.
  /// </summary>
  IToStringConverter ToStringConverter { get; }

  IImageResizer ImageResizer { get; }

  /// <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);

  /// <summary>
  /// Контроллер, отвечающий за взаимодействие различных GUI подсистем внутри
  /// плагина.
  /// </summary>
  IGuiScreensInteractionController InteractionController { get; }
}
```

Свойство **ArchiveEventsReader** предоставляет интерфейс доступа к архиву. Подробные сведение о данном интерфейсе размещены 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 **ShowWindow** должна быть реализована логика работы с пользователем через графический интерфейс. Данный метод вызывается в момент щелчка клавиши мыши (клавиатуры) по пункту меню, связанному с плагином.

Для регистрации плагина **Элемент меню** необходимо на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс **IPlugin**, вызвать метод **RegisterMenuItem** интерфейса хоста (см. Регистрация плагинов в Macroscop).

Плагин Процессор событий

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

```
/// <summary>
/// Класс плагина для обработки событий системы, генерации команд и своих событий
/// </summary>
public abstract class EventProcessor : IDisposable
{
  /// <summary>
  /// Инициализация. Вызывается хостом.
  /// </summary>
  /// <param name="settings">Настройки плагина</param>
  public abstract void Initialize(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 ExecuteCommandSync;

  /// <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);

  public abstract void Dispose();
}
```

Производный класс должен иметь обязательный атрибут **PluginGUINameAttribute**, содержащий название плагина, отображаемое в приложении **Macroscop Конфигуратор**. Если плагин может быть настроена в конфигураторе, следует реализовать метод настройки **SetSettings** и указать атрибут **PluginHasSettingsAttribute**.

В метод инициализации **Initialize** со стороны хоста передается объект настроек плагина. Поскольку каждый плагин прикрепляется пользователем к тому или иному каналу, объект настроек **PluginSettings** состоит из 2 частей:

1) настройки, связанные с текущим каналом безопасности;
2) общие настройки, не зависящие от выбранного канала безопасности.

```
public struct PluginSettings
{
    /// <summary>
    /// Специфические настройки плагина, связанные с текущим каналом.
    /// 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 заполняется объект общих настроек. В случае, когда все настройки плагина привязываются к конкретному каналу, достаточно заполнять только объект специфических настроек канала. Объекты общих и специфических настроек являются пользовательскими (объект настройки канала всегда отличен от null); одним из требованием к ним является возможность сериализации, поскольку все настройки плагина хранятся в общей конфигурации **Macroscop**. Если плагину 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 текущий момент реализованы 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;
- **RawSetDigitalOutputCommand** — устанавливает уровень сигнала на выходе камеры;
- **RawSetDigitalPulsesCommand** — генерирует последовательность импульсов на выходе камеры;

Плагин **Процессор событий** может подписываться на события, возникающие в канале. Для этого на этапе своей инициализации плагин должен заполнить делегат **OnChannelEventReceived**. Кроме того, плагин может генерировать собственные события в канале.

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. Регистрация плагинов в Macroscop).

Плагин Получатель кадров

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

```
/// <summary>
/// Интерфейс получения кадров реального времени.
/// Используется для создания плагинов, получающих кадры
/// с IP-камер.
/// </summary>
public interface ICameraServiceProvider
{
  /// <summary>
  /// Асинхронное получение информации об устройстве
  /// </summary>
  /// <param name="callback"></param>
  /// <param name="state"></param>
  /// <returns></returns>
  IAsyncResult BeginGetCapabilities(AsyncCallback callback, object state);

  /// <summary>
  /// Завершает асинхронный запрос на получение информации об устройстве
  /// </summary>
  /// <param name="asyncResult"></param>
  /// <returns></returns>
  DeviceCapabilities EndGetCapabilities(IAsyncResult asyncResult);

  /// <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>
  /// 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="channelStreamType"></param>
  void StartStream(ChannelStreamTypes channelStreamType);

  /// <summary>
  /// Останавливает поток указаного типа
```

```
  /// </summary>
  /// <param name="channelStreamType"></param>
  void StopStream(ChannelStreamTypes channelStreamType);

  /// <summary>
  /// Отправляет звук на устройство (случай реализации дуплексного звука).
  /// </summary>
  /// <param name="frame"></param>
  void SendSound(RawSoundFrame frame);

  /// <summary>
  /// Интерфейс работы с PTZ
  /// </summary>
  /// <returns></returns>
  IPtzController GetPtzController();

  /// <summary>
  /// Интерфейс работы с цифровыми выходами
  /// </summary>
  /// <returns></returns>
  IDigitalOutputsController GetDigitalOutputsController();

  /// <summary>
  /// Интрейфес для работы с архивом устройства
  /// </summary>
  /// <returns></returns>
  IDeviceArchiveController GetDeviceArchiveController();

  /// <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>
[Flags]
public enum ChannelStreamTypes
{
  /// <summary>
  /// Главный поток видео
  /// </summary>
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.MainVideo")]
  MainVideo = 1,

  /// <summary>
  /// Альтернативный поток видео
  /// </summary>
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.AlternativeVideo")]
  AlternativeVideo = 2,

  /// <summary>
  /// Поток звука, идущий с камеры.
  /// </summary>
```

```
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.MainSound")]
  MainSound = 4,

  /// <summary>
  /// Альтернативный звуковой поток
  /// </summary>
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.AlternativeSound")]
  AlternativeSound = 8,

  /// <summary>
  /// Обратный поток звука.
  /// </summary>
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.OutputSound")]
  OutputSound = 16,

  /// <summary>
  /// Поток данных детекции движения.
  /// </summary>
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.MotionDetection")]
  MotionDetection = 32,

  /// <summary>
  /// Поток данных от системы ввода-вывода камеры
  /// </summary>
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.IO")]
  IO = 64,

  /// <summary>
  /// Архивное видео
  /// </summary>
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.ArchiveVideo")]
  ArchiveVideo = 128,

  /// <summary>
  /// Архивный звук
  /// </summary>
  [Description("MacroscopSDK.IpDevices.ChannelStreamTypes.ArchiveSound")]
  ArchiveSound = 256,
}
```

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 камеры.

Потоки данных **MainSound** и **AlternativeSound**, **OutputSound** состоят из последовательности звуковых кадров **RawSoundFrame**.

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][AlarusSerializable]
public class RawMJPEGFrame : RawVideoFrame
{
  private bool _isNMjpegFrame;

  public bool IsNMjpegFrame
  {
     get { return _isNMjpegFrame; }
     set { _isNMjpegFrame = value; }
  }

  public RawMJPEGFrame()
      {
      }

  public RawMJPEGFrame(IReferencedBytes refBytes)
  {
     UndecodedRefData = refBytes;
  }
}
```

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

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

  }

  /// <summary>
  /// Данные кадра
  /// </summary>
  public RawG711UFrame(IReferencedBytes refBytes)
  {
     UndecodedRefData = refBytes;
     SamplesPerSecond = 8000;
     BitsPerSample = 16;
     Channels = 1;
     Bitrate = 64000;
  }

  /// <summary>
  /// Размер фрэйма должен быть выравнен на размер минимального блока данных,
  /// с которыми работает декодер
  /// </summary>
  /// <returns>Возвращает гранулярность фрэйма</returns>
  public static int Granularity()
  {
     return 80;
  }
}
```

Пример создания MJPEG-кадра доступен в проекте **Camera.csproj**.

В случае если в процессе получения потоков данных произошел обрыв связи с IP- устройством, плагин **Получатель кадров** должен уведомить об этом хост путем генерации события **NoDataConnectionDeviceEvent** с обязательно заполненным полем **StreamTypesMask**, показывающим, в каких потоках данных произошел обрыв соединения.

Для регистрации получателя кадров в системе необходимо заполнить регистрационную информацию устройства **DevType_RegInfo** и регистрационную информацию потока получения данных **MediaStream_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>
  /// Альтернативные идентификаторы устройства, если есть
  /// </summary>
  public Guid[] DeviceAlternativeGuids;

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

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

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

  /// <summary>
  /// Панорамные режимы.
  /// </summary>
  public PanoramicMode[] PanoramicModes;

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

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

  /// <summary>
  /// Получение интерфейса ICameraServiceProvider.
```

```
  /// </summary>
  public GetCameraServiceDelegate GetCameraService;

  /// <summary>
  /// Получение доп. возможностей (доступные разрешения, фпс, кодеки) конкретного
  /// устройства
  /// </summary>
  public GetAdditionalCapabilitiesDelegate GetAdditionalCapabilities;


  /// <summary>
  /// Описание портов, которые можно указывать плагину извне
  /// </summary>
  public List<ExternalNetworkPortDescriptor> ExternalNetworkPortDescriptors;

  /// <summary>
  /// Список поддерживаемых устройством кодеков при передаче звука
  /// </summary>
  public OutputSoundCodec SupportedOutputSoundCodec;

  /// <summary>
  /// Динамическое получение кодека передачи звука. SupportedOutputSoundCodecs
должен
  /// быть выставлен в Dynamic.
  /// </summary>
  public GetOutputSoundCodec GetOutputSoundCodec;

  /// <summary>
  /// Набор инструментов для автопоиска устройства
  /// </summary>
  public GetDiscoveryKitDelegate GetDiscoveryKit;

  /// <summary>
  /// Делегат для смены IP-адреса
  /// </summary>
  public ChangeIpAddressDelegate ChangeIpAddress;

  /// <summary>
  /// Требования для смены IP-адреса
  /// </summary>
  public ChangeIpAddressRequirements ChangeIpAddressRequirements;

  /// <summary>
  /// Тип архива устройства
  /// </summary>
  public DeviceStorageType StorageType;

  /// <summary>
  /// Скорости воспроизведения архива
  /// </summary>
  public double[] ArchivePlaybackSpeeds;

  /// <summary>
  /// Список поддерживаемых аналитиков устройства.
  /// </summary>
  public CameraBuiltInAnalystDescription[] CameraBuiltInAnalystDescriptions;
}
```

Возможности IP-устройства, с которым работает плагин **Получатель кадров**, описываются полем **Capabilities**:

```
/// <summary>
```

```
/// Перечень возможностей устройства
/// </summary>
[Flags]
public enum DevType_Capabilities : ulong
{
  //Коды 512, 1024 и 2048 свободны

  /// <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>
  /// <remarks>
  /// Данный флаг НАДО ВЫСТАВЛЯТЬ, если медиапуть для выполнения CGi-запроса НЕ
  /// СОДЕРЖИТ "mjpeg", "mpeg4", "h264". В противном случае (если флаг
используется),
  /// будет всегда дергаться функция SetCameraSettings у класса для подключения
  /// MJPEG, а у других классов она будет игнорироваться (как например на Axis).
  /// </remarks>
  DeviceParametersFormatIndependent = 8,

  /// <summary>
  /// Устройство может иметь архив
  /// </summary>
  SupportsArchive = 16,

  /// <summary>
  /// Устройство поддерживает воспроизведение архива в обратном направлении
  /// </summary>
  DeviceSupportsBackwardDirectionInArchive = 32,

  /// <summary>
  /// Устройство является видеорегистратором
  /// </summary>
  DeviceIsDvr = 64,

  /// <summary>
  /// Необходимо начинать отсчет каналов в регистраторе или видеосервере с одного,
а
  /// не с нуля как по умолчанию.
  /// </summary>
  ServerStartChannelFromOne = 128,

  /// <summary>
  /// Реализован метод явного получения возможностей устройства
  /// </summary>
  GettingCapabilitiesSupports = 256,

  /// <summary>
  /// Поддержка панорамных камер
```

```
  /// </summary>
  SupportsPanoramicCameras = 4096,

  /// <summary>
  /// Поддержка PTZ
  /// </summary>
  SupportsPtz = 8192,

  /// <summary>
  /// Поддержка цифровых выходов
  /// </summary>
  SupportsDigitalOutputs = 16384,

  /// <summary>
  /// Поддержка омывателя
  /// </summary>
  SupportsWasherAdjusting = 32768,

  /// <summary>
  /// Необходимость использования портов из внешних источников
  /// </summary>
  ExternalNetworkPortsRequired = 65536,

  /// <summary>
  /// Поддержка коррекции видео с SD-карты
  /// </summary>
  SupportsSDVideoCorrection = 131072,

  /// <summary>
  /// Поддержка стабилизации видео с SD-карты
  /// </summary>
  SupportsSDVideoStabilization = 262144,

  /// <summary>
  /// Поддержка множественного доступа к архиву устройства
  /// </summary>
  NotSupportsStorageConcurrentAccess = 524288,

  /// <summary>
  /// Поддержка домофонов
  /// </summary>
  SupportsDoorphone = 1048576,

  /// <summary>
  /// Поддержка работы некоторых функций камеры по защищённому протоколу
  /// </summary>
  SupportsSecureConnection = 2097152
}
```

Если требуется решить задачу управления поворотной камерой или ее выходами, необходимо реализовать соответствующие интерфейсы **IPtzController** и/или **IDigitalOutputsController**:

```
/// <summary>
/// Унифицированный интерфейс реализации управления поворотной камерой.
/// </summary>
public interface IPtzController
{
  /// <summary>
  /// Инициализация камеры.
  /// </summary>
```

```
  /// <returns></returns>
  void Reinitialization();

  /// <summary>
  /// Зачистка ресурсов
  /// </summary>
  void CleanUp();

  /// <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.
  /// <para/>
  /// Если горизонтальная и вертикальная скорость равны 0 - непрерывное движение
  /// будет остановлено.
  /// </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. Скорость равная 0
означает
  /// остановку непрерывного зума.</param>
  void ContiniousZoomIn(int speed);

  /// <summary>
  /// Непрерывное (желательно плавное) отдаление.
  /// </summary>
  /// <param name="speed">Скорость. Интервал от 1 до 100. Скорость равная 0
означает
  /// остановку непрерывного зума.</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();

  /// <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);

/// <summary>
/// Устанавливает автоматическое управление фокусом
/// </summary>
void SetAutoFocus();

/// <summary>
/// Дальний фокус
/// </summary>
/// <param name="step">Шаг 1 до 100.</param>
void FocusFar(int step);

/// <summary>
/// Ближний фокус
/// </summary>
/// <param name="step">Шаг 1 до 100.</param>
void FocusNear(int step);

/// <summary>
/// Непрерывный (желательно плавный) дальний фокус
/// </summary>
/// <param name="speed">Скорость. Интервал от 1 до 100. Если скорость равна 0 -
/// непрерывный фокус будет остановлен.</param>
void ContiniousFocusFar(int speed);

/// <summary>
/// Непрерывный (желательно плавный) ближний фокус
/// </summary>
/// <param name="speed">Скорость. Интервал от 1 до 100. Если скорость равна 0
/// - непрерывный фокус будет остановлен.</param>
void ContiniousFocusNear(int speed);

/// <summary>
/// Устанавливает автоматическое управление диафрагмой
/// </summary>
void SetAutoIris();

/// <summary>
/// Закрывает диафрагму.
/// </summary>
/// <param name="step">Шаг 1 до 100.</param>
void IrisOpen(int step);

/// <summary>
/// Приоткрывает диафрагму.
```

```
  /// </summary>
  /// <param name="step">Шаг 1 до 100.</param>
  void IrisClose(int step);

  /// <summary>
  /// Непрерывное (желательно плавное) открытие диафрагмы.
  /// </summary>
  /// <param name="speed">Скорость. Интервал от 1 до 100. Если скорость равна 0 -
  /// непрерывное открытие диафрагмы будет остановлено.</param>
  void ContiniousIrisOpen(int speed);

  /// <summary>
  /// Непрерывное (желательно плавное) закрытие диафрагмы.
  /// </summary>
  /// <param name="speed">Скорость. Интервал от 1 до 100. Если скорость равна 0 -
  /// непрерывное закрытие диафрагмы будет остановлено.</param>
  void ContiniousIrisClose(int speed);

  /// <summary>
  /// Включить подсветку
  /// </summary>
  void TurnOnInfraredLight();

  /// <summary>
  /// Выключить подсветку
  /// </summary>
  void TurnOffInfraredLight();

  /// <summary>
  /// Запустить стеклоочиститель
  /// </summary>
  void TurnOnWiper();

  /// <summary>
  /// Отстановить стеклоочиститель
  /// </summary>
  void TurnOffWiper();

  /// <summary>
  /// Запустить омыватель
  /// </summary>
  void RunWasher();
}


/// <summary>
/// Уинифицированный интерфейс реализации управления выходами камеры
/// </summary>
public interface IDigitalOutputsController
{
    /// <summary>
    /// Инициализация
    /// </summary>
    /// <returns></returns>
    void Reinitialization();

    /// <summary>
    /// Зачистка ресурсов
    /// </summary>
    void CleanUp();

    /// <summary>
```

```
    /// Возвращает возможности данной камеры.
    /// </summary>
    /// <returns></returns>
    DigitalOutputsCapabilities 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, DigitalImpulse[] pulses);

    /// <summary>
    /// Получить текущие состояния всех выходов
    /// </summary>
    List<DigitalPort> GetOutputsStates();
}
```

Возможности устройства 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**, вызывая метод **RegisterDevType** интерфейса хоста (см. Регистрация плагинов в Macroscop).

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

```
/// <summary>
/// Регистрационная информация потока получения данных
/// </summary>
public class MediaStream_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 MediaStream_Capabilities Capabilities;
}
```

Данную информацию нужно задать столько раз, сколько различных форматов потока поддерживает IP-устройство.

Аналогично **DevType_RegInfo**, информацию **MediaStream_RegInfo** необходимо задавать
на этапе загрузки сборки с плагином в методе инициализации класса, реализующего интерфейс **IPlugin**. Для этого достаточно вызвать метод **RegisterMediaStreamInfo** интерфейса хоста.

Пример заполнения данных классов доступен в проекте **Camera.csproj**.