

www.satvision-cctv.ru

SATVISION smart
systems
система профессионального IP-видеонаблюдения

Satvision Smart Systems

SDK

Оглавление

1. Общие сведения	3
2. Плагины	4
2.1 Зависимости	4
2.2 Подключение.....	4
2.3 Интерфейсы	4
2.3.1 IPlugin	4
2.3.2 IApiContext	5
2.3.3 IEventSubscriber.....	7
2.3.4 IServerHealthSubscriber.....	8
2.3.5 IExternalAction.....	8
2.3.6 IExternalActionEditControl.....	8
2.3.7 IExternalModule	9
2.3.8 IExternalModuleEditControl	9
2.3.9 IExternalVideoDetector	9
2.3.10 IHttpRequestResponder	10
2.4 Примеры	10
3. Интеграция	15
3.1 Зависимости	15
3.2 Использование	15
3.3 Классы	16
3.3.1 ConnectionWrapper	16
3.3.2 LifeViewWrapper	17
3.3.3 AudioPlayer.....	18
3.3.4 RSVideoWriterWrapper	18
3.3.5 VideoDecoderWrapper.....	19
3.3.6 VideoRendererWrapper	19
3.3.7 RealtimeVideoControl	20
3.3.8 ArchiveVideoControl.....	21
3.4 Примеры	24

1. Общие сведения

Satvision Smart Systems SDK – это инструментарий, позволяющий:

- Создавать программное обеспечение, именуемое плагинами, позволяющее расширять существующие функциональные возможности программного обеспечения Satvision Smart Systems.
- Интегрировать функциональные возможности Satvision Smart Systems в стороннее программное обеспечение.

Для работы с Satvision Smart Systems SDK требуется наличие установленных на ПК следующих программных компонентов:

- Microsoft Visual Studio 2012 или выше;
- Microsoft .Net Framework 4.5 Full или выше.

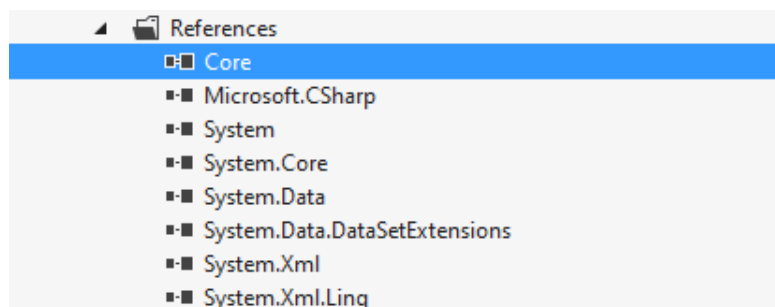
При использовании Satvision Smart Systems SDK требуется соблюдать зависимости библиотек (см. пункты «Зависимости»).

SDK состоит из библиотек, требуемых для разработки и примеров, демонстрирующих решение различных задач.

2. Плагины

2.1 Зависимости

Для реализации плагинов требуется добавить в ваш проект ссылку на библиотеку Core.dll. Библиотека находится в директории Dependencies пакета SDK.



2.2 Подключение

Функционал библиотеки Core.dll располагается в namespace «Satvision Smart Systems.VMS». Собранная сборка помещается в директорию "C:\Program Files\Satvision Smart Systems\modules". При старте сервер/клиент ищет и загружает сборки из данной директории.

2.3 Интерфейсы

2.3.1 IPlugin

IPlugin – центральный интерфейс внешнего подключаемого модуля. Идентификатор модуля (поле ID) должен быть постоянным и уникальным для класса, его реализующего. При старте сервер/клиент ищет в сборке классы, реализующие данный интерфейс и инициализирует их (вызывает функцию Initialize). Функция Initialize, используя переданный контекст, выполняет необходимые настройки (подписку на события, регистрацию внешних действий и т.д.). Метод Free вызывается перед закрытием сервера/клиента и позволяет корректно освободить задействованные ресурсы.

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

    /// <summary>
    /// Возвращает понятное пользователю название модуля.

```

```

    /// </summary>
    string Name { get; }

    /// <summary>
    /// Возвращает описание модуля.
    /// </summary>
    string Description { get; }

    /// <summary>
    /// Вызывается сервером для инициализации модуля.
    /// </summary>
    void Initialize(IApiContext context);

    /// <summary>
    /// Вызывается сервером перед отключением модуля.
    /// </summary>
    void Free();
}

```

2.3.2 IApiContext

IApiContext – контекст доступа к функционалу сервера видеонаблюдения. Все взаимодействие – получение объектов, подписка на события и тд. производится через данный класс. Экземпляр класса, реализующего данный интерфейс, передается в метод Initialize плагина и может быть сохранен и использован в течение всего цикла выполнения программы.

```

public interface IApiContext
{
    /// <summary>
    /// Позволяет зарегистрировать внешнее действие сценария.
    /// </summary>
    /// <remarks>
    /// Действие должно быть помечено атрибутом SerializableAttribute.</remarks>
    /// <exception cref="System.ArgumentNullException"/>
    /// <exception cref="System.SystemException"/>
    void RegisterScriptAction(Type actionType);

    /// <summary>
    /// Позволяет зарегистрировать внешний модуль аналитики.
    /// </summary>
    /// <remarks>
    /// Объект должен быть помечен атрибутом SerializableAttribute.</remarks>
    /// <exception cref="System.ArgumentNullException"/>
    /// <exception cref="System.SystemException"/>
    void RegisterModule(Guid moduleId, string moduleName, byte licenseCount, Type moduleType);

    /// <summary>
    /// Регистрирует подписчика на системные события.
    /// </summary>
    void RegisterEventSubscriber(IEventSubscriber subscriber);

    /// <summary>
    /// Отменяет регистрацию подписчика на системные события.
    /// </summary>
    void UnRegisterEventSubscriber(IEventSubscriber subscriber);

    /// <summary>
    /// Регистрирует подписчика на состояние сервера.

```

```

/// </summary>
void RegisterServerHealthSubscriber(IServerHealthSubscriber subscriber);

/// <summary>
/// Отменяет регистрацию подписчика на состояние сервера.
/// </summary>
void UnRegisterServerHealthSubscriber(IServerHealthSubscriber subscriber);

/// <summary>
/// Регистрирует обработчика HTTP-запросов.
/// </summary>
void RegisterHttpRequestResponder(IHttpRequestResponder responder);

/// <summary>
/// Отменяет регистрацию обработчика HTTP-запросов.
/// </summary>
void UnRegisterHttpRequestResponder(IHttpRequestResponder responder);

/// <summary>
/// Удаляет карточки модуля FR, входящие в заданную группу.
/// </summary>
void DeleteFRCardsByGroupName(string groupName);

/// <summary>
/// Получение информации о сервере.
/// </summary>
ServerInfo GetServerInfo();

/// <summary>
/// Возвращает список объектов конфигурации заданного типа.
/// </summary>
/// <exception cref="System.ArgumentNullException"/>
List<T> GetConfigurationObjects<T>() where T : class;

/// <summary>
/// Возвращает объект конфигурации заданного типа по его идентификатору.
/// </summary>
/// <exception cref="System.ArgumentNullException"/>
T GetConfigurationObject<T>(Guid id) where T : class, IObjekt;

/// <summary>
/// Возвращает объект конфигурации заданного типа по его имени.
/// </summary>
/// <exception cref="System.ArgumentNullException"/>
T GetConfigurationObject<T>(string name) where T : class, INamed;

/// <summary>
/// Сохраняет объект конфигурации.
/// </summary>
/// <exception cref="System.ArgumentNullException"/>
/// <exception cref="System.SystemException"/>
void ChangeConfigurationObject(BaseObject obj);

/// <summary>
/// Удаляет объект конфигурации.
/// </summary>
/// <exception cref="System.ArgumentNullException"/>
/// <exception cref="System.SystemException"/>
void DeleteConfigurationObject(BaseObject obj);

/// <summary>
/// Возвращает текущее состояние счетчика посетителей для заданной камеры.
/// </summary>
CounterAnalyticsResult GetCounterState(Guid cameraId);

```

```

    /// <summary>
    /// Возвращает текущее состояние FR-аналитики посетителей для заданной камеры в виде
    /// списка данных различного типа.
    /// </summary>
    List<FRAnalyticsResult> GetFRCounterStates(Guid cameraId);

    /// <summary>
    /// Генерирует событие.
    /// </summary>
    void RaiseEvent(Guid cameraId, CameraAlarmLevel level, string message, string
description);

    /// <summary>
    /// Создает объект управления PTZ для заданной камеры.
    /// </summary>
    /// <exception cref="System.InvalidOperationException"/>
    /// <exception cref="System.SystemException"/>
    IPtzControl CreatePtzControl();

    /// <summary>
    /// Запуск форсированной записи на заданной камере.
    /// </summary>
    bool StartForcedRecord(Guid cameraId, int lengthInSec);

    /// <summary>
    /// Остановка форсированной записи на заданной камере.
    /// </summary>
    bool StopForcedRecord(Guid cameraId);

    /// <summary>
    /// Временное переключение записи с дополнительного потока на основной.
    /// </summary>
    bool SwitchToBaseStream(Guid cameraId, int lengthInSec);

    /// <summary>
    /// Временное отключение прореживания видеопотока.
    /// </summary>
    bool DisableFrameSkipping(Guid cameraId, int lengthInSec);
}

```

2.3.3 IEventSubscriber

IEventSubscriber – интерфейс обработчика событий сервера. Для получения событий класс, реализующий данный интерфейс, должен быть зарегистрирован через метод RegisterEventSubscriber контекста IApiContext.

```

public interface IEventSubscriber
{
    /// <summary>
    /// Обработывает событие.
    /// </summary>
    /// <exception cref="System.ArgumentNullException"/>
    /// <exception cref="System.SystemException"/>
    void Process(CameraEvent cameraEvent);
}

```

2.3.4 IServerHealthSubscriber

IServerHealthSubscriber – интерфейс обработчика состояния сервера. Для получения данных класс, реализующий данный интерфейс, должен быть зарегистрирован через метод RegisterServerHealthSubscriber контекста IApiContext.

```
public interface IServerHealthSubscriber
{
    /// <summary>
    /// Обрабатывает текущее состояние сервера.
    /// </summary>
    /// <exception cref="System.ArgumentNullException"/>
    /// <exception cref="System.SystemException"/>
    void Process(ServerHealth serverHealth);
}
```

2.3.5 IExternalAction

IExternalAction – интерфейс внешнего действия, которое будет доступно в разделах «Сценарии» и «Расписания» в Satvision Smart Systems.

```
public interface IExternalAction : INamed
{
    /// <summary>
    /// Выполняет действие, используя заданный контекст.
    /// </summary>
    /// <exception cref="System.ArgumentNullException"/>
    /// <exception cref="System.SystemException"/>
    void Execute(ICameraScriptContext context);

    /// <summary>
    /// Возвращает элемент управления, позволяющий редактировать действие, либо null, если редактирование не требуется.
    /// Элемент управления должен быть унаследован от System.Windows.Control.
    /// </summary>
    IExternalActionEditControl GetEditControl();
}
```

2.3.6 IExternalActionEditControl

IExternalActionEditControl – интерфейс элемента управления (должен быть унаследован от System.Windows.Forms.Control) для редактирования внешних действий.

```
public interface IExternalActionEditControl
{
    /// <summary>
    /// Установка действия для редактирования.
    /// </summary>
    void SetValue(IExternalAction action);

    /// <summary>
    /// Возвращает отредактированное действие, либо null, если в данный момент невозможно получить значение
    /// </summary>
    IExternalAction GetValue();
}
```

2.3.7 IExternalModule

IExternalModule – интерфейс внешнего модуля аналитики, который будет доступен для включения в списках модулей в окне редактирования камеры.

```
public interface IExternalModule
{
    /// <summary>
    /// Создает объект, выполняющий анализ видеопотока.
    /// </summary>
    /// <exception cref="System.ArgumentNullException"/>
    /// <exception cref="System.SystemException"/>
    IExternalVideoDetector CreateVideoDetector(IApiContext context, Guid cameraId);

    /// <summary>
    /// Возвращает элемент управления, позволяющий редактировать модуль, либо null, если
    редактирование не требуется.
    /// Элемент управления должен быть унаследован от System.Windows.Control.
    /// </summary>
    IExternalModuleEditControl GetEditControl();
}
```

2.3.8 IExternalModuleEditControl

IExternalModuleEditControl – интерфейс элемента управления (должен быть унаследован от System.Windows.Forms.Control) для редактирования настроек модуля аналитики.

```
public interface IExternalModuleEditControl
{
    /// <summary>
    /// Установка модуля для редактирования настроек.
    /// </summary>
    void SetValue(IExternalModule module);

    /// <summary>
    /// Возвращает отредактированный модуль, либо null, если в данный момент невозможно
    получить значение.
    /// </summary>
    IExternalModule GetValue();
}
```

2.3.9 IExternalVideoDetector

IExternalVideoDetector – интерфейс видеодетектора для внешнего модуля видеоаналитики.

```
public interface IExternalVideoDetector : IDisposable
{
    /// <summary>
    /// Выполняет анализ, возвращая результат.
    /// </summary>
    void PerformAnalysis(Bitmap bitmap, DateTime bitmapTime, DateTime lastMotionTime);

    /// <summary>
    /// Возвращает результат анализа, либо null, если результатов в данный момент нет.
    /// </summary>
    ExternalAnalyticsResult GetResult();
}
```

2.3.10 IHttpRequestResponder

IHttpRequestResponder – интерфейс внешнего обработчика HTTP-запросов.

```
public interface IHttpRequestResponder
{
    /// <summary>
    /// Возвращает true, если запрос по данному URI может быть обработан.
    /// </summary>
    bool CanProcess(bool url);

    /// <summary>
    /// Обрабатывает запрос.
    /// </summary>
    /// <exception cref="System.ArgumentNullException"/>
    /// <exception cref="System.SystemException"/>
    void GetResult(HttpListenerContext context, ManualResetEvent cancelEvent);
}
```

2.4 Примеры

Реализация интерфейса IPlugin.

```
using System;

namespace Sample
{
    class Plugin : IPlugin
    {
        private object _syncRoot = new object();
        private List<IDisposable> _dependentObjects = new List<IDisposable>();

        public Guid ID
        {
            get { return new Guid("{37FC3E2D-B8B9-43FD-89E5-487E2CC93978}"); }
        }

        public string Name
        {
            get { return "Event example"; }
        }

        public string Description
        {
            get { return "This plugin demonstrates how to subscribe to events and how generate them"; }
        }

        public void Initialize(IApiContext context)
        {
            // Мы подписываемся на системные события и обрабатываем их так, как нам нужно
            context.RegisterEventSubscriber(new SampleEventSubscriber(context));

            // Мы можем генерировать события, когда нам это нужно
            // Давайте сохраним генератор событий в списках зависимых объектов, чтобы мы могли
            // избавиться от него позже
            // Нам нужно использовать блокировку, потому что методы этого плагина могут
            // вызываться из разных потоков
            var eventGenerator = new SampleEventGenerator(context);
            eventGenerator.Start();
        }
    }
}
```

```

        lock (_syncRoot)
        {
            _dependentObjects.Add(eventGenerator);
        }
    }

    public void Free()
    {
        lock (_syncRoot)
        {
            _dependentObjects.ForEach(obj => obj.Dispose());
            _dependentObjects.Clear();
        }
    }
}
}

```

Пример генерации событий.

SampleEventGenerator.cs

```

using System;
using System.IO;
using System.Timers;
using Revisor.VMS;

namespace ExternalScriptActionExample
{
    [Serializable]
    class SampleEventGenerator : IDisposable
    {
        #region Properties

        private IApiContext _context;
        private object _syncRoot = new object();
        private Timer _timer;

        #endregion

        #region Methods

        /// <summary>
        /// Создает экземпляр типа SampleEventGenerator.
        /// </summary>
        /// <exception cref="System.ArgumentNullException">
        public SampleEventGenerator(IApiContext context)
        {
            if (context == null)
            {
                throw new ArgumentNullException();
            }

            _context = context;
        }

        /// <summary>
        /// Запускает генерацию событий.
        /// </summary>
        public void Start()
        {
            lock (_syncRoot)
            {
                if (_timer == null)
                {

```

```

        _timer = new Timer(5000);
        _timer.Elapsed += timer_Elapsed;
        _timer.Start();
    }
}

/// <summary>
/// Уничтожает текущий объект.
/// </summary>
public void Dispose()
{
    lock (_syncRoot)
    {
        if (_timer != null)
        {
            _timer.Dispose();
            _timer = null;
        }
    }
}

private void timer_Elapsed(object sender, ElapsedEventArgs e)
{
    lock (_syncRoot)
    {
        if (_context != null)
        {
            _context.RaiseEvent(Guid.Empty, CameraAlarmLevel.Notification, "Timer
event", "5000 ms elapsed");
        }
    }
}

#endregion
}
}

```

Получение событий с записью информации о них в текстовый файл.

SampleEventSubscriber.cs

```

using System;
using System.IO;
using Revisor.VMS;

namespace ExternalScriptActionExample
{
    [Serializable]
    class SampleEventSubscriber : IEventSubscriber
    {
        #region Properties

        private IApiContext _context;

        #endregion
        #region Methods

        /// <summary>
        /// Создает экземпляр типа SampleEventSubscriber.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"/>
        public SampleEventSubscriber(IApiContext context)
        {

```

```

        if (context == null)
        {
            throw new ArgumentNullException();
        }

        _context = context;
    }

    /// <summary>
    /// Обрабатывает событие.
    /// </summary>
    /// <exception cref="System.ArgumentNullException"/>
    /// <exception cref="System.SystemException"/>
    public void Process(CameraEvent cameraEvent)
    {
        if (cameraEvent == null)
        {
            throw new ArgumentNullException();
        }

        // Давайте запишем все тревожные события в текстовый файл
        if (cameraEvent.Level == CameraAlarmLevel.Alarm)
        {
            WriteToSystemAlarmLogSafe(cameraEvent);
        }

        // Мы можем использовать команды API для выполнения действий
        // Давайте представим, что нам нужно переключить нашу камеру "CameraA" на
        // предустановку PTZ по умолчанию при обнаружении движения на "CameraB".
        if (cameraEvent.CameraID != Guid.Empty)
        {
            var cameraA = _context.GetConfigurationObject<Camera>("CameraA");
            var cameraB = _context.GetConfigurationObject<Camera>("CameraA");

            if (cameraA != null && cameraB != null && cameraEvent.CameraID ==
cameraB.ID)
            {
                var ptz = _context.CreatePtzControl();
                ptz.DoAdvancedOperation(cameraA.ID, PtzAdvancedOperation.CallPreset,
1);

                ptz.Close();
            }
        }

        private void WriteToSystemAlarmLogSafe(CameraEvent cameraEvent)
        {
            try
            {
                var fileName = Path.Combine(
                    Environment.GetFolderPath(Environment.SpecialFolder.Desktop,
System.Environment.SpecialFolderOption.None),
                    "test_system_events.txt");

                File.AppendAllText(fileName, string.Format("System event: time={0},
type={1}\r\n", cameraEvent.Time.ToString(), cameraEvent.Type.ToUserString()));
            }
            catch (Exception ex)
            {
                DebugLog.Errors.WriteError("Error while writing to file.", ex);
            }
        }

        #endregion
    }

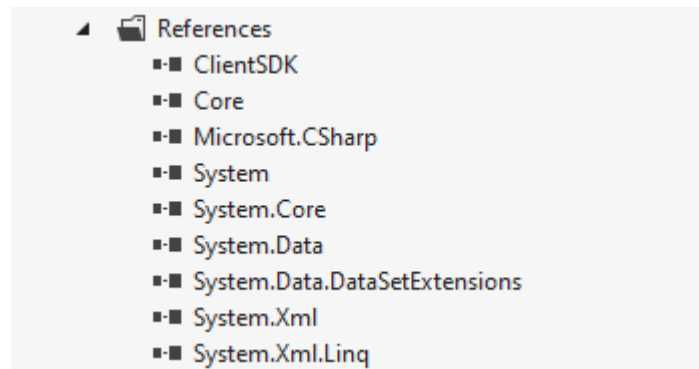
```

} }

3. Интеграция

3.1 Зависимости

Для интеграции в стороннее программное обеспечение требуется добавить в ваш проект ссылки на библиотеки Core.dll и ClientSDK.dll. В проекте необходимо использовать namespace Satvision Smart Systems.VMS и namespace Satvision Smart Systems.VMS.ClientSDK



Библиотека ClientSDK.dll зависит от библиотек из поставки Satvision Smart Systems:

Multimedia.dll, Client.dll и библиотек, расположенных в директориях x86 и amd64. Весь необходимый набор библиотек находится в директории Dependencies пакета SDK.

Библиотеки Client.dll и Multimedia.dll необходимо вручную скопировать в директорию сборки проекта. Если приложение взаимодействует с видео/аудио (проигрывание реалтайм/архивного видео или распаковка видео/аудио), то требуется также скопировать директории x86 и amd64 со всеми библиотеками в директорию сборки проекта.

3.2 Использование

После подключения библиотек и указания требуемых пространств имен вы можете добавлять в ваш проект требуемые классы, методы, UserControl'ы.

Общий список классов:

- AudioPlayer – проигрыватель аудио.
- ConnectionWrapper – удаленное подключение к серверу видеонаблюдения.
- LifeViewWrapper – позволяет получать видео с камеры в режиме реального времени.
- RSVIDWriterWrapper – позволяет осуществлять экспорт видео в файл в формате RSVID.
- VideoDecoderWrapper – декодер видео.
- VideoRendererWrapper – рендерер видео.

- RealtimeVideoControl – элемент управления, позволяющий проигрывать видео с камеры в режиме реального времени.
- ArchiveVideoControl – элемент управления, позволяющий проигрывать видео из архива.

3.3 Классы

3.3.1 ConnectionWrapper

Класс, который инициализирует удаленное подключение к серверу видеонаблюдения.

Свойства

```
/// <summary>
/// Возвращает true, если подключение установлено.
/// </summary>
public bool Connected { get; }
```

Методы:

```
/// <summary>
/// Создает экземпляр типа ConnectionWrapper.
/// </summary>
public ConnectionWrapper()

/// <summary>
/// Выполняет подключение к серверу.
/// </summary>
public void Connect(string serverAddress, int serverPort, string userName, string
password, bool useHttps, int timeout)

/// <summary>
/// Выполняет асинхронное подключение к серверу.
/// </summary>
public async Task ConnectAsync(string serverAddress, int serverPort, string userName,
string password, bool useHttps = false, int timeout = 30000)

/// <summary>
/// Возвращает информацию о сервере.
/// </summary>
public ServerInfo GetServerInfo()

/// <summary>
/// Возвращает серверную конфигурацию.
/// </summary>
public ServerConfiguration GetConfiguration()

/// <summary>
/// Возвращает серверную конфигурацию в формате, поддерживаемом COM.
/// </summary>
public ComServerConfiguration GetComConfiguration()

/// <summary>
/// Добавление объекта.
/// </summary>
public void AddObject(BaseObject obj)

/// <summary>
/// Изменение объекта.
/// </summary>
public void ChangeObject(BaseObject obj)
```

```

/// <summary>
/// Удаление заданного объекта.
/// </summary>
public void DeleteObject(BaseObject obj)

/// <summary>
/// Запускает форсированную запись для заданной камеры.
/// </summary>
public void StartForcedRecord(Guid cameraId, int lengthInSec)

/// <summary>
/// Останавливает форсированную запись для заданной камеры.
/// </summary>
public void StopForcedRecord(Guid cameraId)

/// <summary>
/// Создает объект LifeView, используя актуальный метод передачи данных.
/// </summary>
public LifeViewWrapper CreateLifeView(Camera camera, bool baseSource, int fps, bool
useTestView)

/// <summary>
/// Загружает записи заданной камеры за период.
/// </summary>
public List<CameraSequence> GetCameraSequences(Guid cameraId, DateTime timeFrom, DateTime
timeTo)

/// <summary>
/// Освобождает ресурсы, занятые объектом.
/// </summary>
public void Dispose()

```

3.3.2 LifeViewWrapper

Класс, позволяющий получать видео с камеры в режиме реального времени.

Методы

```

/// <summary>
/// Создает экземпляр типа LifeViewWrapper.
/// </summary>
public LifeViewWrapper(IReferenceLifeView lifeView)

/// <summary>
/// Получение текущего кадра.
/// </summary>
public Frame GetFrame()

/// <summary>
/// Возвращает текст текущей ошибки, либо пустую строку, если ошибки нет.
/// </summary>
public string GetErrorText()

/// <summary>
/// Обновляет индекс желаемого потока.
/// </summary>
public void UpdateSource(bool baseSource)

/// <summary>
/// Закрывает подключение к камере.
/// </summary>
public void Close()

/// <summary>
/// Освобождает ресурсы, занятые объектом.

```

```
/// </summary>
public void Dispose()
```

3.3.3 AudioPlayer

Класс, который позволяет проигрывать аудио.

Методы

```
/// <summary>
/// Создает экземпляр типа AudioPlayerHelper.
/// </summary>
public AudioPlayerWrapper()

/// <summary>
/// Добавляет заданный аудиокادر в очередь асинхронного проигрывания.
/// </summary>
public bool PlayAsyncSafe(AudioFrame frame)

/// <summary>
/// Добавляет заданные аудиокадры в очередь асинхронного проигрывания.
/// </summary>
public bool PlayAsyncSafe(List<AudioFrame> frames)

/// <summary>
/// Очищает очередь асинхронного проигрывания.
/// </summary>
public void Clear()

/// <summary>
/// Освобождает ресурсы, занятые объектом.
/// </summary>
public void Dispose()
```

3.3.4 RSVideoWriterWrapper

Класс, который позволяет осуществлять экспорт видео в файл в формате RSVID (специальный формат, исключающий перекодирование при сохранении данных).

Методы

```
/// <summary>
/// Создает экземпляр типа RSVideoWriterWrapper.
/// </summary>
public RSVideoWriterWrapper(string fileName)
    : this(fileName, null)

/// <summary>
/// Создает экземпляр типа RSVideoWriterWrapper.
/// </summary>
public RSVideoWriterWrapper(string fileName, List<CameraSequence> sequences)

/// <summary>
/// Производит запись кадра в файл.
/// </summary>
public void Write(Frame frame)

/// <summary>
/// Освобождает ресурсы, занятые объектом.
/// </summary>
public void Dispose()
```

3.3.5 VideoDecoderWrapper

Класс, отвечающий за декодирование видео.

Методы

```
/// <summary>
/// Создает экземпляр типа VideoDecoderWrapper.
/// </summary>
public VideoDecoderWrapper()

/// <summary>
/// Распаковывает заданный кадр.
/// </summary>
public DecodedVideoFrame Decode(VideoFrame frame)

/// <summary>
/// Освобождает ресурсы, занятые объектом.
/// </summary>
public void Dispose()
```

3.3.6 VideoRendererWrapper

Класс, отвечающий за рендер видео.

Методы

```
/// <summary>
/// Создает экземпляр типа VideoRendererWrapper.
/// </summary>
public VideoRendererWrapper()

/// <summary>
/// Выполняет подготовку видеокadra для вывода в область заданного размера:
/// масштабирование, затем конвертирование в RGB32.
/// </summary>
public bool Render(DecodedVideoFrame decodedFrame, Size renderSize, bool convertToRGB32 =
true)

/// <summary>
/// Выполняет копирование отрендеренного изображения в заданный битмап.
/// Если размеры битмапа отличаются от размеров рендеринга, то производится
/// автоматическое масштабирование.
/// </summary>
public bool CopyRendered(Bitmap bitmap)

/// <summary>
/// Кодирует изображение в формат JPEG.
/// </summary>
public byte[] CompressRenderedToJpeg()

/// <summary>
/// Возвращает формат подготовленного к рендерингу изображения.
/// </summary>
public DecodedVideoFormat GetRenderFormat()

/// <summary>
/// Возвращает размеры подготовленного к рендерингу изображения.
/// </summary>
public Size GetRenderSize()

/// <summary>
/// Освобождает ресурсы, занятые объектом.
/// </summary>
public void Dispose()
```

3.3.7 RealtimeVideoControl

Элемент управления, позволяющий проигрывать видео с камеры в режиме реального времени.

Свойства

```
/// <summary>
/// Возвращает, либо устанавливает значение, определяющее, что проигрывание видео стоит
/// на паузе.
/// </summary>
public bool IsFreezed { get; set; }

/// <summary>
/// Возвращает, либо устанавливает значение, определяющее необходимость проигрывания
/// звука.
/// </summary>
public bool PlayAudio { get; set; }

/// <summary>
/// Возвращает камеру, либо null, если проигрывание не ведется.
/// </summary>
public Camera Camera { get; }

/// <summary>
/// Возвращает, либо устанавливает тип проигрываемого потока (Auto/Main/Second).
/// Значение по умолчанию: VideoSourceType.Auto.
/// </summary>
public VideoSourceType SourceType { get; set; }

/// <summary>
/// Возвращает, либо устанавливает значение, определяющее вид масштабирования.
/// Значение по умолчанию: ImageStretchType.WithRatio.
/// </summary>
public ImageStretchType StretchType { get; set; }
```

События

```
/// <summary>
/// Событие получения кадра элементом управления (вызывается асинхронно).
/// </summary>
public event FrameReceiveEventHandler FrameReceive;
```

Методы

```
/// <summary>
/// Создает экземпляр типа RealtimeVideoControl.
/// </summary>
public RealtimeVideoControl()

/// <summary>
/// Запуск проигрывания.
/// </summary>
public bool Play(Camera camera, int fps)

/// <summary>
/// Остановка проигрывания.
/// </summary>
public void Stop()

/// <summary>
/// Возвращает копию изображения, отображаемого в данный момент, либо null, если
/// проигрывание не ведется.
/// </summary>
```

```
public Bitmap GetCurrentImageCopy()
```

3.3.8 ArchiveVideoControl

Элемент управления, позволяющий проигрывать видео из архива.

Свойства

```
/// <summary>
/// Возвращает, или устанавливает значение, определяющее проигрывать ли аудио.
/// </summary>
public bool PlayAudio { get; set; }

/// <summary>
/// Определяет, отображать или нет шкалу времени.
/// Значение по умолчанию: true.
/// </summary>
public bool ShowDiagram { get; set; }

/// <summary>
/// Определяет, отображать или нет панель управления.
/// Значение по умолчанию: true.
/// </summary>
public bool ShowToolBar { get; set; }

/// <summary>
/// Возвращает true, если идет проигрывание.
/// </summary>
public bool IsPlaying { get; }

/// <summary>
/// Возвращает идентификатор выбранной камеры.
/// </summary>
public Guid SelectedCameraID { get; }

/// <summary>
/// Возвращает true, если для камеры, отображаемой в ячейке, можно включить проигрывание звука.
/// </summary>
public bool CanPlayAudio { get; }

/// <summary>
/// Возвращает список видеорядов.
/// </summary>
public List<CameraSequence> Sequences { get; }

/// <summary>
/// Возвращает, либо устанавливает значение, определяющее вид масштабирования.
/// Значение по умолчанию: ImageStretchType.WithRatio.
/// </summary>
public ImageStretchType StretchType { get; set; }

/// <summary>
/// Возвращает, либо устанавливает скорость проигрывания.
/// Значение по умолчанию: 1.00.
/// Возможные значения: от 0.20 до 64.0.
/// </summary>
public double Speed { get; set; }
```

События

```
/// <summary>
/// Событие изменения состояния проигрывателя.
/// </summary>
```

```
public event EventHandler StateChanged;
```

Методы

```
/// <summary>
/// Создает экземпляр типа ArchiveVideoControl.
/// </summary>
public ArchiveVideoControl()

/// <summary>
/// Производит инициализацию проигрывателя.
/// </summary>
public void InitNow(Guid cameraId)

/// <summary>
/// Производит инициализацию проигрывателя.
/// </summary>
public void InitTime(Guid cameraId, DateTime time, PolygonF[] regionsOfInterest = null,
bool autoPlay = false)

/// <summary>
/// Увеличение скорости проигрывания.
/// </summary>
public void IncSpeed()

/// <summary>
/// Уменьшение скорости проигрывания.
/// </summary>
public void DecSpeed()

/// <summary>
/// Выбор камеры.
/// </summary>
public bool SelectCamera()

/// <summary>
/// Очистка камеры.
/// </summary>
public void ClearCamera()

/// <summary>
/// Обновление.
/// </summary>
public void ReInit()

/// <summary>
/// Проигрывание вперед.
/// </summary>
public void PlayForward()

/// <summary>
/// Проигрывание назад.
/// </summary>
public void PlayBackward()

/// <summary>
/// Пауза.
/// </summary>
public void Pause()

/// <summary>
/// Переход к следующему видеоряду.
/// </summary>
public void NextSequence()
```

```

/// <summary>
/// Переход к предыдущему видеоряду.
/// </summary>
public void PreviousSequence()

/// <summary>
/// Переход к следующему кадру.
/// </summary>
public void NextFrame()

/// <summary>
/// Переход к предыдущему кадру.
/// </summary>
public void PreviousFrame()

/// <summary>
/// Возвращает копию изображения, отображаемого в данный момент.
/// </summary>
public Bitmap GetCurrentImageCopy()

/// <summary>
/// Возвращает размер выводимого в данный момент изображения.
/// </summary>
public Size GetCurrentImageSize()

/// <summary>
/// Возвращает текущее время.
/// </summary>
public DateTime GetCurrentTime()

/// <summary>
/// Экспортирует видео.
/// </summary>
public void ExportVideo()

/// <summary>
/// Прои́грывание вперед/пауза.
/// </summary>
public void ToggleForward()

/// <summary>
/// Прои́грывание назад/пауза.
/// </summary>
public void ToggleBackward()

/// <summary>
/// Прибли́жение изображения.
/// </summary>
public void ZoomIn()

/// <summary>
/// Отдале́ние изображения.
/// </summary>
public void ZoomOut()

/// <summary>
/// Сбро́с масштабирования.
/// </summary>
public void ResetZoom()

/// <summary>
/// Возвращает масштаб изображения.
/// </summary>

```

```

public float GetZoom()

/// <summary>
/// Отображает окно перехода к дате/времени.
/// </summary>
public void GoToDateTime()

/// <summary>
/// Перегружает текст, выводимый при отсутствии выбранной камеры.
/// </summary>
public void OverrideEmptyText(string emptyText)

/// <summary>
/// Выполняет действие над камерой FishEye.
/// </summary>
public void DoFishEyeAction(FishEyeAction action)

/// <summary>
/// Возвращает параметры FishEye.
/// </summary>
public FishEyeParams GetFishEyeParams()

/// <summary>
/// Устанавливает параметры FishEye.
/// </summary>
public void SetFishEyeParams(FishEyeParams renderParams)

```

3.4 Примеры

Добавление пользователя в систему видеонаблюдения.

```

static void Main(string[] args)
{
    try
    {
        using (var connection = new ConnectionWrapper())
        {
            connection.Connect("localhost", 11012, "admin", "");
            Console.WriteLine("Connected.");

            var serverInfo = connection.GetServerInfo();
            Console.WriteLine("Got server info. Server version: " +
serverInfo.Version.ToString());

            var configuration = connection.GetConfiguration();
            Console.WriteLine("Got server configuration. User count: " +
configuration.Users.Count);

            const string TEST_USER_NAME = "TestUser";
            var testUser = configuration.Users.Find(u => u.Name.Equals(TEST_USER_NAME));

            if (testUser != null)
            {
                Console.WriteLine(TEST_USER_NAME + " already exists");
            }
            else
            {
                testUser = new User {
                    ID = Guid.NewGuid(),
                    Name = TEST_USER_NAME,
                    Description = "Demonstration of SDK capabilities",
                };

                testUser.Rights = AccessRights.CreateDefaultUserRights();
                //testUser.Rights.SetRole(SecurityRole.Administration, true);
            }
        }
    }
}

```

```

        connection.AddObject(testUser);

        Console.WriteLine(TEST_USER_NAME + " is successfully added");
    }

    Console.WriteLine("Press Enter to exit");
    Console.ReadLine();
}
catch (Exception ex)
{
    Console.WriteLine("ERROR: " + ex.Message);
}
}

```

Проигрывание видеопотока в реальном времени с использованием RealtimeVideoControl.

RecordHelper.cs

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Revisor.VMS;
using Revisor.VMS.ClientSDK;

namespace ViewRealtimeExample
{
    class RecordHelper : IDisposable
    {
        #region Properties

        private RealtimeVideoControl _videoControl;
        private RSVIDWriterWrapper _writer;
        private string _fileName;
        private object _syncRoot = new object();
        private Exception _exception;

        /// <summary>
        /// Возвращает имя файла.
        /// </summary>
        public string FileName
        {
            get { return _fileName; }
        }

        #endregion

        #region Methods

        /// <summary>
        /// Создает экземпляр типа RecordHelper.
        /// </summary>
        /// <exception cref="System.ArgumentNullException">
        public RecordHelper(RealtimeVideoControl videoControl)
        {
            if (videoControl == null)
            {
                throw new ArgumentNullException();
            }

            _videoControl = videoControl;

```

```

        _videoControl.FrameReceive += videoControl_FrameReceive;

        _fileName = Path.Combine(
            Directory.GetCurrentDirectory(),
            string.Format("{0}.rsvid",
                DateTime.Now.ToString("yyyy_MM-dd-HH-mm-ss",
CultureInfo.InvariantCulture)));

        _writer = new RSVIDWriterWrapper(_fileName);
    }

    /// <summary>
    /// Возвращает ошибку, либо null, если ошибок не было.
    /// </summary>
    public Exception GetErrorIfOccured()
    {
        return _exception;
    }

    /// <summary>
    /// Освобождает ресурсы, занятые объектом.
    /// </summary>
    public void Dispose()
    {
        lock (_syncRoot)
        {
            if (_videoControl != null)
            {
                _videoControl.FrameReceive -= videoControl_FrameReceive;
                _videoControl = null;
            }

            Kernel.DisposeObjectSafe(ref _writer);
        }
    }

    private void videoControl_FrameReceive(object sender, FrameReceiveEventArgs args)
    {
        lock (_syncRoot)
        {
            try
            {
                if (_writer != null)
                {
                    _writer.Write(args.Frame);
                }
            }
            catch (Exception ex)
            {
                _exception = ex;
                Kernel.DisposeObjectSafe(ref _writer);
            }
        }
    }

    #endregion
}

```

TestForm.cs

```

using System;
using System.Drawing.Imaging;
using System.Globalization;

```

```

using System.IO;
using System.Windows.Forms;
using Revisor.VMS.ClientSDK;

namespace ViewRealtimeExample
{
    public partial class TestForm : Form
    {
        #region Properties

        private ConnectionStatus _status;
        private ConnectionWrapper _connection;
        private RecordHelper _recordHelper;

        private enum ConnectionStatus
        {
            Disconnected,
            Connecting,
            Connected
        }

        #endregion
        #region Methods

        /// <summary>
        /// Создает экземпляр типа TestForm.
        /// </summary>
        public TestForm()
        {
            InitializeComponent();
            SetStatus(ConnectionStatus.Disconnected);
        }

        private async void StartConnect(string serverName, string user, string password)
        {
            SetStatus(ConnectionStatus.Connecting);

            try
            {
                _connection = new ConnectionWrapper();
                await _connection.ConnectAsync(serverName, 11012, user, password);
                FinishConnect(null);
            }
            catch (Exception ex)
            {
                FinishConnect(ex);
            }
        }

        private void FinishConnect(Exception ex)
        {
            if (this.InvokeRequired)
            {
                this.Invoke(new Action<Exception>(FinishConnect), ex);
                return;
            }

            if (_connection != null && _connection.Connected)
            {
                SetStatus(ConnectionStatus.Connected);
                InitiatePlayback();
            }
            else
            {

```

```

        CancelConnect();

        if (ex != null)
        {
            MessageBox.Show("ERROR: " + GetFullExceptionText(ex));
        }
    }

private void InitiatePlayback()
{
    if (_connection != null)
    {
        var configuration = _connection.GetConfiguration();
        var cameras = configuration.Cameras;
        var camera = cameras.Find(c => !c.Disabled);

        videoControl.Play(camera, 30);
    }
}

private void CancelConnect()
{
    videoControl.Stop();

    if (_connection != null)
    {
        _connection.Dispose();
        _connection = null;
    }

    SetStatus(ConnectionStatus.Disconnected);
}

private void SetStatus(ConnectionStatus status)
{
    _status = status;

    labelStatus.Text = _status.ToString();
    textServer.Enabled = textUser.Enabled = textPassword.Enabled = (_status ==
ConnectionStatus.Disconnected);

    buttonConnect.Text = (_status == ConnectionStatus.Disconnected
        ? "Connect"
        : "Disconnect");

    buttonStreamAuto.Enabled = (_status == ConnectionStatus.Connected);
    buttonFreeze.Enabled = (_status == ConnectionStatus.Connected);
    buttonAudio.Enabled = (_status == ConnectionStatus.Connected);
    buttonSnapshot.Enabled = (_status == ConnectionStatus.Connected);
    buttonRecord.Enabled = (_status == ConnectionStatus.Connected);

    if (_status != ConnectionStatus.Connected)
    {
        buttonFreeze.Checked = false;
        buttonAudio.Checked = false;
        buttonRecord.Checked = false;

        StartOrStopRecord(false);
    }
}

private string GetFullExceptionText(Exception ex)
{

```

```

        var result = ex.Message;
        var innerEx = ex.InnerException;

        while (innerEx != null)
        {
            result += Environment.NewLine + innerEx.Message;
            innerEx = innerEx.InnerException;
        }

        return result;
    }

    private void StartOrStopRecord(bool enable)
    {
        if (enable && _recordHelper == null)
        {
            _recordHelper = new RecordHelper(videoControl);
        }
        else if (!enable && _recordHelper != null)
        {
            var fileName = _recordHelper.FileName;
            var exception = _recordHelper.GetErrorIfOccured();

            if (exception != null)
            {
                MessageBox.Show("Recording saved with errors. File name: "
                    + _recordHelper.FileName
                    + Environment.NewLine
                    + "Error: "
                    + exception.Message);
            }
            else
            {
                MessageBox.Show("Recording saved to: " + _recordHelper.FileName);
            }

            _recordHelper.Dispose();
            _recordHelper = null;
        }

        buttonRecord.Checked = enable;
    }

    private void buttonConnect_Click(object sender, EventArgs e)
    {
        if (_status == ConnectionStatus.Disconnected)
        {
            StartConnect(textServer.Text, textUser.Text, textPassword.Text);
        }
        else
        {
            CancelConnect();
        }
    }

    private void buttonFreeze_Click(object sender, EventArgs e)
    {
        videoControl.IsFreezed = buttonFreeze.Checked;
    }

    private void buttonAudio_Click(object sender, EventArgs e)
    {
        videoControl.PlayAudio = buttonAudio.Checked;
    }

```

```

private void buttonSnapshot_Click(object sender, EventArgs e)
{
    try
    {
        using (var bitmap = videoControl.GetCurrentImageCopy())
        {
            if (bitmap == null)
            {
                MessageBox.Show("Image is missing");
                return;
            }

            var fileName = Path.Combine(
                Directory.GetCurrentDirectory(),
                string.Format("{0}.jpeg",
                    DateTime.Now.ToString("yyyy_MM-dd-HH-mm-ss",
CultureInfo.InvariantCulture)));

            bitmap.Save(fileName, ImageFormat.Jpeg);

            MessageBox.Show("Saved to " + fileName);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("ERROR: " + GetFullExceptionText(ex));
    }
}

private void buttonStreamAuto_Click(object sender, EventArgs e)
{
    videoControl.BaseSource = null;
    buttonStreamAuto.Checked = true;
    buttonStreamFirst.Checked = false;
    buttonStreamSecond.Checked = false;
}

private void buttonStreamFirst_Click(object sender, EventArgs e)
{
    videoControl.BaseSource = true;
    buttonStreamAuto.Checked = false;
    buttonStreamFirst.Checked = true;
    buttonStreamSecond.Checked = false;
}

private void buttonStreamSecond_Click(object sender, EventArgs e)
{
    videoControl.BaseSource = false;
    buttonStreamAuto.Checked = false;
    buttonStreamFirst.Checked = false;
    buttonStreamSecond.Checked = true;
}

private void buttonRecord_Click(object sender, EventArgs e)
{
    StartOrStopRecord(!buttonRecord.Checked);
}

private void TestForm_FormClosed(object sender, FormClosedEventArgs e)
{
    StartOrStopRecord(false);
    CancelConnect();
}

```

```
}  
    }  
#endregion
```